El presente proyecto fin de carrera consiste en el diseño, desarrollo e implementación de una aplicación informática cuya función sea la identificación de distintos ficheros de imagen, audio y video y la interpretación y presentación de los metadatos asociados a los mismos.

El software desarrollado, EXTRACTORDATOS_LBS, reconocerá el tipo de formato del fichero bajo estudio a partir del análisis de los bytes de identificación contenidos en la cabecera del archivo. En base a la información registrada en dicha cabecera, la aplicación interpretará el contenido de los metadatos asociados al fichero, mostrando por pantalla aquellos que resulten de interés para el análisis de los mismos.

Previamente a la implementación del software se acomete el análisis teórico de los formatos de diversos archivos multimedia, recogidos en múltiples normas y recomendaciones. Tras esa identificación, se procede al desarrollo de la aplicación EXTRACTORDATOS_LBS , que informa de los parámetros de interés contenidos en las cabeceras de los archivos. El desarrollo se ilustra con los diagramas conceptuales asociados a la arquitectura del software implementado. De igual forma, se muestran las salidas por pantalla de una serie de ficheros de muestra, y se presenta el manual de usuario de la aplicación. La versión electrónica de este documento acompaña el ejecutable que permite el análisis de los archivos.

This final project consists in the design, development and implementation of a computer application whose function is the identification of different image, audio and video files and the interpretation and presentation of their metadata.

The software developed, EXTRACTORDATOS_LBS, will recognize the type of the file under study through the analysis of the identification bytes contained on the file's header. Based on information registered in this header, the application will interpret the metadata content associated to file, displaying the most interesting ones for their analysis.

Prior to the software implementation, a theoretical analysis of the different formats of media files is undertaken. After this identification, the application EXTRACTORDATOS_LBS is developed. This software analyzes and displays the most interesting parameters contained in multimedia file's header. The development of the application is illustrated with flow charts associated to the architecture of the software.

Furthermore, some graphic examples of use of the program are included, as well as the user's manual. The electronic version of this document attaches the executable file that permits file analysis.

Proyecto Fin de Carrera

Software para análisis automático de ficheros de imagen

Lucía Berrocal Sáez

Septiembre 2012

GLOSARIO

GIF: Graphics Interchange Format

PNG: Portable Network Graphics

BMP: Windows Bitmap

PGM: Portable Graymap

PPM: Portable Pixmap

PBM: Portable Bitmap

TIFF: Tagged Image File Format

XPM: X Pixmap

PCX: Personal Computer exchange

PSD: Photoshop Document

DPX: Digital Picture exchange

JPEG: Joint Photographic Expert Group

MIDI: Music Instrument Digital Interface

WAV: Waveform Audio File Format

FLAC: Free Lossless Audio Codec

AIFF: Audio Interchange File Format

WMA: Windows Media Audio

MPEG: Moving Picture Expert Group

AVI: Audio Video Interleave

WMV: Windows Media Video

FLV: Flash Video

INDICE

INDICE DE ILUSTRACIONES Y TABLAS

1. INTRODUCCIÓN

Desde que la tecnología analógica comenzó a dar paso a la tecnología digital en lo que a imagen, audio y vídeo se refiere, muchos han sido los tipos de formatos multimedia que han ido apareciendo.

Debido a condiciones de copyright, variantes de compresión o normas y estándares, a día de hoy podemos encontrar que una misma foto, por ejemplo, puede ser almacenada en un dispositivo electrónico con muchos formatos de archivo distintos, dando lugar a distintas estructuras de almacenamiento de una misma información.

Ante esta situación, el departamento de criminalística de la Guardia Civil se vio en la necesidad de implementar un software que mostrara las particularidades más importantes de los archivos multimedia más utilizados a día de hoy, dado que aunque existan programas de este tipo en la red, no están lo suficientemente autentificados para dichas pericias judiciales. El programa desarrollado, del que es objeto el presente proyecto fin de carrera, será muy útil para investigaciones forenses de archivos audiovisuales.

1.1. ¿Qué se pretende con este proyecto?

Para poder encontrar los datos de interés que ofrecen los ficheros digitales, este proyecto consiste en la creación de un software programado en lenguaje C++ que sea capaz de identificar el tipo de fichero analizado y a continuación mostrar, si procede, todos los datos que vaya a ofrecer dicho archivo.

1.2. Estructura de la memoria

La presente memoria se estructurará principalmente en una importante introducción teórica de formatos de imagen, vídeo y audio, una descripción de la programación del software con explicaciones, diagramas de flujo y partes de código y un sencillo manual de usuario.

1.3. ¿Qué es C++?

Como se ha mencionado anteriormente, este software ha sido programado en el lenguaje orientado a objetos C++, con el programa de Microsoft Visual Studio 2010.

C++ es un lenguaje de programación que nació en 1980 [1] con la intención de extender el lenguaje C a la manipulación de objetos. A día de hoy se puede usar para una programación genérica, orientada a objetos o estructurada, es decir, puede usarse para la programación orientada a objetos o bien para la programación no orientada a objetos.[2]

1.4. Programación orientada a objetos (OOP)

La programación orientada a objetos no es un lenguaje específico sino un método de programar. Con ella se pretende trabajar con la misma filosofía con la que se utilizan la mayoría de los objetos que nos rodean hoy en día; se les pide "órdenes" y sin saber cómo funcionan internamente se obtiene el resultado deseado. Gracias a este método, se ahorra la división en partes de un problema que hay que llevar a cabo a la hora de programar una posible solución. Para programar de esta manera, es necesario que el lenguaje lo soporte, para ello es necesario que incorpore clases con las características de OOP.

1.5. Librerias MFC

Las librerías MFC son una lista de funciones predeterminadas que ofrece Microsoft [3]. Por lo general son funciones básicas que simplifican código y facilitan la programación. Ejemplos de estas funciones podrían ser leer un fichero en hexadecimal, almacenar variable en un buffer, realizar una suma, etc.

2. BASE TEÓRICA

Antes de introducirse plenamente en el proyecto como tal, es necesario explicar cómo es la estructura de un archivo digital, dado que será el objeto con el que se trabajará exhaustivamente.

Los archivos digitales se dividen en dos partes importantes, la cabecera (header) y el resto de información. La cabecera alberga todos los datos que nosotros queremos conocer, como por ejemplo el tamaño, el número de colores de la imagen, la duración de un vídeo, la frecuencia de muestreo de un archivo de audio, etcétera.

Las cabeceras se encuentran siempre al inicio del fichero y son una serie de caracteres alfanuméricos. La primera parte de la cabecera es conocida como "números mágicos" [4] o bytes de identificación y, por lo general, estos caracteres ocupan 4 bytes (aunque veremos más adelante que no todos los archivos lo cumplen). Seguido de los bytes de identificación, encontraremos el resto de información requerida. Dependiendo del tipo de formato que se considere, la información estará almacenada en una posición u otra y de diferentes maneras, lo que hace fundamental la identificación del formato mediante los bytes de identificación, o números mágicos, para el posterior análisis del archivo.

Esta información se encontrará principalmente de las siguientes formas [5]:

- BIG ENDIAN: Los bytes de mayor a menor peso se encuentran de izquierda a derecha.
- LITTLE ENDIAN: Al contrario que el anterior, se ordenan de derecha a izquierda.
- VALORES EN ASCII: Los datos vendrán directamente impresos en código ASCII.
- VALORES EQUIVALENTES: Serán números con equivalencia a un significado. Por ejemplo 1= Codificación RLE

A continuación se enumeran los tipos de archivo de imagen, audio y vídeo que el programa podrá estudiar. A parte de estos formatos, habrá otros que el programa será capaz de reconocer pero no de leer sus datos dado que se trata de formatos obsoletos o que no se ajustan a lo que se pretende analizar con este software. Los formatos admitidos son:

- PNG
- BMP
- PPM
- PGM
- PBM
- PCX
- GIF
- PSD
- DPX
- JPEG
- MP3
- MIDI
- WAV
- FLAC
- MPEG-1
- MPEG-2
- AVI
- AIFF
- FLV

Y reconocerá el formato pero no analizará:

- TIFF

- RGB
- XPM
- PM
- MKV
- RM
- OGG
- MOV
- MP4
- M4V

2.1. Formatos de imagen

Los archivos de imagen tienen dos modos principales para ser manipulados. Estos modos son las imágenes de mapa de bits y las imágenes vectoriales.

2.1.1. Generalidades

2.1.1.1. Imágenes de mapa de bits

Las imágenes de mapa de bits [6] están formadas por una rejilla de celdas. Cada celda (píxel) tiene asignado un color y un valor de luminancia consiguiendo así una sensación de tonos continuos.

Los píxeles no son de un tamaño concreto, por tanto, dependiendo del número de celdas que se elijan para una imagen, se tendrá mayor o menor resolución, siendo mayor cuanto más píxeles tengamos.

Cuando modificamos el tamaño de la imagen, también cambiamos los valores asignados de los píxeles, produciéndose así deformaciones dentro de la imagen.

Considerando esto último, las imágenes de mapa de bits tienen un tamaño determinado y siempre que se modifique se perderá calidad.

2.1.1.2. Imágenes vectoriales

El peso de las imágenes vectoriales es mucho más pequeño que el de las imágenes de mapa de bits ya que ordenan la información de una manera más sencilla.

Se generan objetos de la imagen mediante cálculos matemáticos [7]. De esta manera, se visualizan los gráficos vectoriales a partir de las coordenadas de unas líneas que se guardan como referencia.

A diferencia de los mapas de bits, cada objeto de la imagen puede modificarse individualmente sin alterar los demás. Además, si se modifica el tamaño no se pierde calidad ya que el propio ordenador recalcula las coordenadas y los vectores para que la imagen guarde la proporción original.

2.1.1.3. Tipos de compresión

Debido a que algunos archivos de imagen ocupan mucho espacio, ralentizando así el manejo en ordenadores comunes, se han desarrollado diversas técnicas de compresión de imágenes que consiguen reducir notablemente el volumen de dichos archivos, disminuyendo así recursos y tiempos de transferencia.

Existen varios métodos de compresión, tanto de dominio público como de empresas que los desarrollan. Se dividen principalmente en compresión con pérdida y compresión sin pérdida.

A primera vista se podría suponer que es mejor no tener pérdidas a tenerlas, pero ahora veremos que según para qué queramos comprimir una imagen, será mejor un método u otro.

2.1.1.3.1. Algoritmos de compresión sin pérdida

Los algortimos de compresión sin pérdida condensan el código sin despreciar nada de la información que contiene la imagen para posteriormente poder recuperarla a la

perfección [8]. Obviamente, el nivel de compresión que ofrecerá este método es mucho más bajo que el método con pérdida pero es el óptimo si lo que se quiere es una exacta visualización de la imagen.

Entre los métodos de compresión sin pérdida tenemos:

- RLE (Run Length Encoded)

Este algoritmo analiza la imagen y determina los píxeles que son del mismo color [9]. Básicamente se sustituye una determinada secuencia de bits por un código, habiéndose registrado antes el color y la posición de dicho color en la imagen. Con imágenes con muchas zonas del mismo color se consigue un alto nivel de compresión, ocurriendo todo lo contrario cuando la imagen tiene gran cantidad de tonos, obteniendo a veces mayor peso que la imagen original.

Se utiliza principalmente en imágenes de barrido.

- LZW (Lemple-Zif-Welch)

Es muy parecido al algoritmo anterior con la diferencia de que éste es utilizado por muchos más formatos de imagen como pdf, gif, o tif.

- HUFFMAN

La codificación Huffman es un método de compresión que se basa en probabilidad estimada que tiene un símbolo de aparecer [10].

Basándose en eso, se crea un código específico para cada símbolo, siendo de menor tamaño para los símbolos con más frecuencia de aparición y viceversa.

El método Huffman es idóneo para la codificación símbolo a símbolo.

- ZIP

Este método se utiliza para comprimir cualquier tipo de archivo.

2.1.1.3.2. Algorítmos de compresión con pérdida

Los algoritmos de compresión sin pérdida eliminan información de las imágenes para poder comprimirlas mejor. A veces se eliminan gamas de colores en la transición de uno a otro o se suavizan los bordes.

- JPEG

Este método se basa en dos cualidades que tiene el ojo humano: la capacidad de apreciar mejor un cambio de brillo frente a un cambio de color y el poder distinguir con mayor claridad cambios de brillo en una zona homogénea que en una no homogénea. Aprovechándose de esto, se eliminan las altas frecuencias de la imagen sin afectar apenas al resultado. La compresión jpeg es variable, consiguiendo así el grado de compresión que quiera el usuario perdiendo más o menos información [11].

2.1.2. Tipos de archivo

2.1.2.1. GIF

Es un formato de imagen que se usa principalmente en la World Wide Web para imágenes y animaciones.

Se hizo muy famoso porque utilizaba el algoritmo de compresión LZW, capaz de alcanzar un alto grado de compresión y reducir el tiempo de descarga frente al algoritmo RLE.

El formato GIF no tiene pérdidas siempre y cuando las imágenes tengan 256 colores como mucho. Si tuvieran más, la imagen se tendría que adaptar eliminando colores, lo que sí supondría pérdida de calidad.

Debido a que el algoritmo estaba patentado, todas las compañías que querían usar imágenes GIF tenían que pagar a Unisys (la compañía propietaria de los derechos). Por ello, el formato PNG comenzó a tomar más relevancia frente al GIF.

Los bytes de identificación del formato GIF son:

Posición	Bytes
0	47
1	49
2	46
3	38

Tabla 1 Bytes de identificación de formato GIF

Y su estructura de datos [12]:

Posición	**Tamaño (bytes)**	**Significado**
0	3	“GIF”
3	3	“87a” o “89a”
6	2	Anchura
8	2	Altura
10	1	Bit 0: Global color Table Flag Bit 1..3: Color resolution bit 4: Sort Flag to Global Color Table bit 5..7: Size of Global Color Table: 2^(1+n)
11	1	Color del background
12	1	Radio del pixel

Tabla 2 Estructura del formato GIF

Todos los bytes están en orden Little-endian.

Los bytes 3,4y 5 especifican la versión del formato. 87a se refiere a la primera versión, creada en 1987, y 89a es la actualización más reciente que soporta varias imágenes en el archivo y animación.

El byte con posición 10 no se tendrá en cuenta dado que no ofrecen datos de nuestro interés.

2.1.2.2. PNG

Debido a la política que tomó Unisys de cobrar la patente del algoritmo LZW que utiliza el formato GIF, los usuarios comenzaron a utilizar el formato PNG (Portable Network Graphics). Se creó como alternativa al formato GIF no cobrando patente y mejorando los fallos que tenía el anterior formato como por ejemplo que solo soportara 256 colores cuando ya existían ordenadores que soportaban millones.

PNG se basa en un algoritmo de compresión sin pérdida para mapas de bits que reduce el tamaño del archivo, manteniendo la calidad de la imagen.

Sin embargo, en general los ficheros PNG ocupan bastante más que los JPEG o los GIF, además, los PNG no pueden hacer animaciones por lo que se sigue usando el formato GIF para ello.

Los bytes de identificación del formato PNG son:

Posición	Bytes
0	89
1	50
2	4E
3	47

Tabla 3 Bytes de identificación de formato PNG

Y su estructura de datos es la siguiente [13]:

Posición	**Tamaño (bytes)**	**Significado**
16	4	Anchura
20	4	Altura

24	1	Profundidad de bit
25	1	Tipo de color
26	1	Método de compresión
27	1	Tipo de filtro
28	1	Método de entrelazado

Tabla 4 Estructura formato PNG

Tanto la altura como la anchura de la imagen como la profundidad de bits vienen escritos en big-endian.

La altura y la anchura expresan las dimensiones en pixeles y la profundidad de bits el número de bits por muestra.

El tipo de color nos describe la interpretación de los datos de la imagen. Los valores pueden ser 0, 2, 3, 4 o 6 y cada valor tiene el significado que se muestra en la siguiente tabla:

Tipo de color	**Profundidad de bit permitida**	**Interpretación**
0	1,2,4,8,16	Cada pixel es una muestra en escala de grises
2	8, 16	Cada pixel es un R,G,B triple
3	1, 2, 4, 8	Cada pixel es un indice de la paleta. El chunk PLTE siempre debe aparecer
4	8, 16	Cada pixel es una muestra en escala de grises seguido por una muestra alpha

6	8, 16	Cada pixel es un R,G,B triple seguido por una muestra alpha

Tabla 5 Tipo de color para formato PNG

Respecto a la compresión, ya se ha mencionado que a día de hoy el formato PNG no se comprime pero, por si acaso, se dejó un byte para futuras actualizaciones. En este momento, dicho byte siempre será 0.

El formato PNG acepta los siguientes filtros:

Tipo	**Nombre**
0	None
1	Sub
2	Up
3	Average
4	Paeth

Tabla 6 Filtros para formato PNG

El filtro **Sub** transmite la diferencia entre un byte y el byte correspondiente al pixel anterior.

El filtro **Up** es igual que el filtro Sub excepto que el píxel inmediatamente por encima del píxel actual, en lugar de a su izquierda, se utiliza como el predictor.

El filtro **Average** hace la media entre los dos pixeles vecinos de la izquierda y de arriba y predice el valor del pixel.

El filtro **Paeth** hace una función lineal simple con los tres pixeles colindantes izquierdo, arriba y arriba a la izquierda y elige el pixel colindante más parecido al valor obtenido.

Respecto al entrelazado existe la opción de implementarlo o no. El método de entrelazado que se emplea es el Adam7 que consiste en siete pasadas distintas sobre una imagen. Cada pasada transmite un subconjunto de pixeles de la imagen. La pasada en la que se envía cada pixel se define mediante la copia del siguiente patrón 8 por 8 para toda la imagen, comenzando desde la esquina superior izquierda.

2.1.2.3. BMP

Es un formato de mapa de bits. Teóricamente permite compresión en imágenes de 4 y 8 bits con la compresión RLE (Run-lenght encoding) aunque nunca se utiliza, guardando siempre las imágenes sin comprimir y por tanto, con gran peso pero con muy alta calidad. Puede guardar imágenes de hasta 24 bits [14].

El formato BMP se utiliza tanto en PC como en MAC, siendo muy usando en programas como Word o Excel.

Debido a su gran tamaño, las imágenes BMP sólo son soportadas por Internet Explorer, siendo muy habitual que se transformen a ficheros JPEG, GIF o PNG para poder comprimirse.

Los bytes de identificación son:

Posición	Bytes
0	42
1	4D

Tabla 7 Bytes de identificación de formato BMP

Y la estructura:

Posición	**Tamaño (bytes)**	**Significado**
14	4	El tamaño de esta cabecera (40 bytes)
18	4	Anchura en pixels (signo integrado).

22	4	Altura en pixels (signo integrado).
26	2	Número de planos de color utilizados
28	2	Número de bits por pixel
30	4	Tipos de compresión (mirar siguiente tabla)
34	4	El tamaño de la imagen. Este es el tamaño de los datos de mapa de bits en bruto. No debe confundirse con el tamaño del archivo.
38	4	Resolución horizontal (pixels por metro)
42	4	Resolución vertical (pixels por metro)
46	4	El número de colores en la paleta de colores, de 0 por defecto a 2^n

Tabla 8 Estructura de formato BMP

Todos los valores en bytes de este formato vienen dados en Little-endian.

La compresión por otro lado, viene dada por una tabla de equivalencias siendo cada valor un tipo de compresión distinto.

A día de hoy solo existen seis tipos de compresión para este formato, pudiendo ser suficiente expresarlo con un byte pero se han reservado cuatro bytes para versiones futuras.

Los posibles tipos de compresión son los siguientes:

Valor	Identificado por:	Método de compresión
0	BI_RGB	Ninguno
1	BI_RLE8	RLE 8-bit/pixel
2	BI_RLE4	RLE 4-bit/pixel
3	BI_BITFIELDS	Bit field o compresión Huffman 1D por BITMAPCOREHEADER2
4	BI_JPEG	JPEG o RLE-24 compresión por BITMAPCOREHEADER2
5	BI_PNG	PNG
6	BI_ALPHABITFIELDS	Bit field

Tabla 9 Tipos de compresión formato BMP

Donde **BI_RGB** es el más común, **BI_RLE8** y **BI-RLE4** indican que sólo se pueden usar en mapas de bits de 8 o 4 bits/pixel respectivamente. **BI_JPEG** y **BI_PNG** significan que el mapa de bit contiene una imagen **JPEG** o **PNG** y **BI_ALPHABITFIELDS** que es válido en Windows CE .NET 4.0 o superior.

2.1.2.4. PGM, PPM y PBM

Son formatos de gráficos simples que suelen contener texto plano y se modifican con editores de texto como por ejemplo el blog de notas de Windows.

El formato PGM se representa en escala de grises utilizando 8 o 16 bits por píxel. PPM es igual pero en color y el PBM en blanco y negro utilizando un bit por píxel.

Sus bytes de identificación en hexadecimal son los siguientes [15]:

	PPM	o también	PBM	o también	PGM	o también
Posición	Bytes	Bytes	Bytes	Bytes	Bytes	Bytes
0	50	50	50	50	50	50
1	36	51	31	34	32	35

Tabla 10 Bytes identificación formatos PPM, PBM y PGM

Su información viene dada directamente en Ascii con lo cual se leerá de la misma forma que se lee el texto. A veces, aparecen comentarios entre los bytes de identificación y los datos de información, siempre señalizados con el carácter # y espacio. Los comentarios no tienen una longitud determinada, ni un número limitado de cuántos poner, por eso, se ha editado una función específica para analizar si el fichero lleva comentarios o no para poder identificar el resto de valores sin dificultad.

La estructura de estos formatos es realmente simple, siempre que no haya comentarios, los dos primeros bytes son los bytes identificadores seguidos de un espacio en blanco, la anchura de la imagen, espacio en blanco, altura, espacio en blanco y máximo de colores (excepto en los PBM que no tienen colores).

Para explicar mejor la teoría de los comentarios, se muestra a continuación un ejemplo del software ya implementado, con un formato PGM con comentario y sin comentario dónde se podrá ver la diferencia entre ellos:

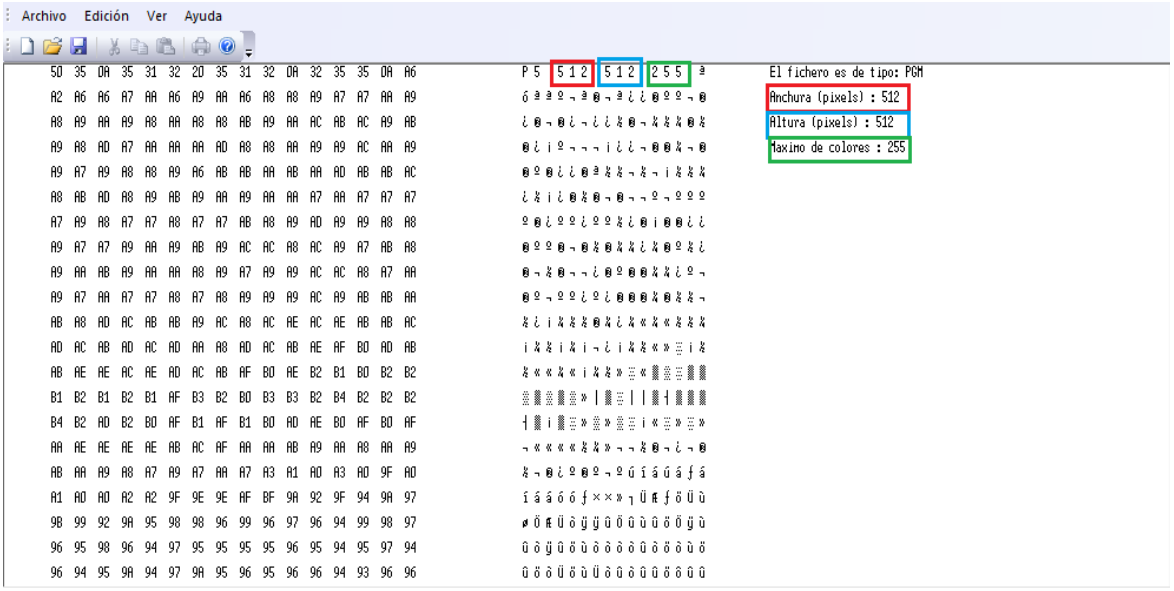


Ilustración 1 Ejemplo de archivo sin comentarios

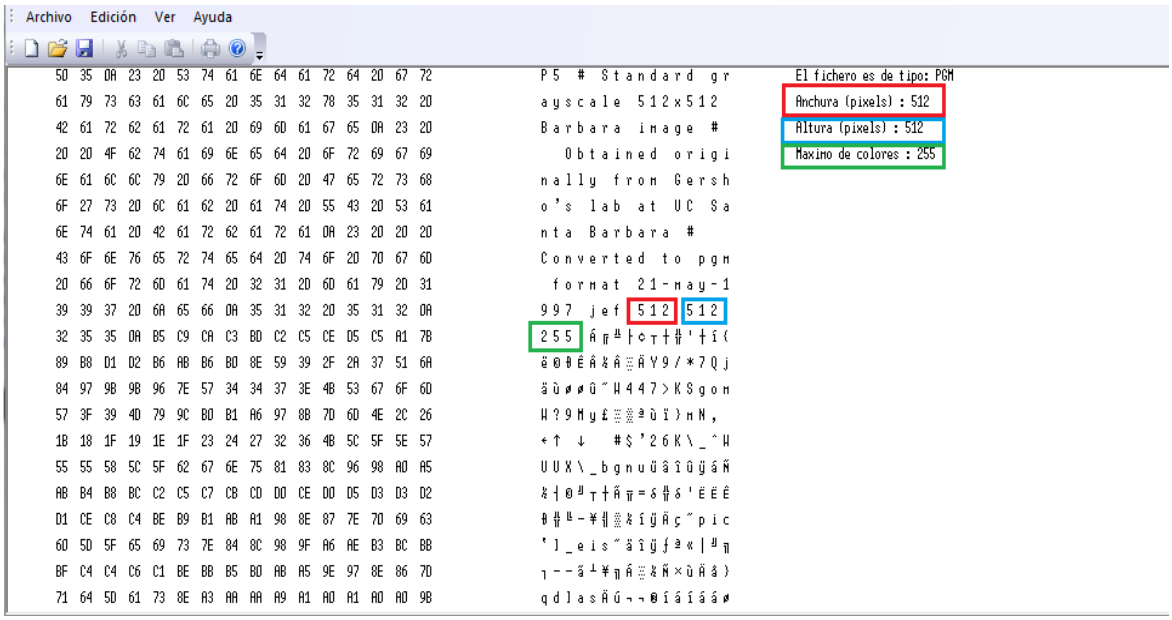


Ilustración 2 Ejemplo archivo con comentarios

2.1.2.5. TIFF

Es un formato de archivo de imágenes de mapa de bits que suele representar imágenes que provienen de escáneres, capturadores de fotogramas o programas de retoque fotográfico.

Los datos en un formato TIFF pueden venir expresados en el orden llamado Intel (lo que se ha explicado como Little-endian) u orden Motorola (Big-endian). Para saber cuál de los dos es el que estamos estudiando, los bytes de identificación son [16]:

	Motorola	Intel
Posición	Bytes	Bytes
0	4D	49
1	4D	49
2	2A	2A

Tabla 11 Bytes de identificación de formato TIFF

Los dos primeros bytes los podríamos traducir en ASCII como MM para el orden Motorola e II para el Intel.

Dado que como se ha comentado, es un formato de capturadores y escáneres, no se incidirá mucho más en el estudio de la estructura de este formato.

2.1.2.6. XPM

Almacena los datos de la imagen en ASCII en una cadena de caracteres estándar de C. Almacenándolos así, se consigue que las imágenes puedan ser modificadas con cualquier editor de texto.

Este tipo de ficheros no puede ser comprimido.

Este formato es poco usado hoy en día por su obsoleta organización de datos, por ello, el software no lo analizará en profundidad pero sí reconocerá el formato con los bytes de identificación siguientes:

Posición	Bytes
0	2F
1	2A
2	20
3	58
4	D
5	A
6	1A
7	A

Tabla 12 Bytes identificación de formato XPM

2.1.2.7. PCX

ZSoft Corporation lo desarrolló para el programa de PC Paintbrush.

Utiliza la compresión RLE y además utiliza un algoritmo que ocupa poca memoria que se encarga de ordenar la paleta de colores poniendo los más usados al principio de ésta, siendo más fáciles de comprimir los primeros que los últimos.

Fue un formato muy utilizado por usuarios de Windows y DOS pero ya ha sido reemplazado por otros formatos de compresión mejores como pueden ser JPEG o PNG.

Su byte de identificación es [17]:

Posición	Bytes
0	0A

Tabla 13 Bytes de identificación de formato PCX

Y la estructura de sus datos ordenados en Little-endian es:

Posición	**Bytes**	**Significado**
0	1	Identificación: Debe ser 10 (0AH) = Archivo PCX
1	1	Versión del fichero PCX: 0= Versión 2.5 2= Versión 2.8 con Paleta 3= Versión 2.8 con Paleta por defecto 4= Paintbrush para Windows 5= Versión 3.0 o superior
2	1	Codificación Debe ser 1= RLE
3	1	Bits por Pixel 1 - Monocromo 4 - 16 colores 8 – 256 colores 24 – 16-7 millones de colores o truecolor
4	8	Coordenadas de imagen Xmin, Ymin, Xmax (el ancho será Xmax-Xmin+1 y el alto será Ymax-Ymin+1)
12	2	Resolución horizontal en ppp (puntos por pulgada)
14	2	Resolución vertical en ppp (puntos por pulgada)
16	48	Mapa de colores con la definición de la paleta, si es una imagen de 16 colores o menos. Organizado en campos bytes de 16*3

64	1	Reservado
65	1	Cantidad de planos, max. 4
66	2	Bytes por línea de imagen (el ancho de la imagen en bytes)
68	2	Información de la paleta 1=color 2=escala de grises
70	2	Anchura del dispositivo en pixeles (a partir de Paintbrush IV)
72	2	Altura del dispositivo en píxeles (a partir de Paintbrush IV)
74	54	Bytes de relleno (hasta completar los 128 bytes).

Tabla 14 Estructura del formato PCX

2.1.2.8. PSD

Es el formato del programa Adobe Photoshop. Guarda los archivos con 48 bits por píxel aparte de almacenar también, las capas, canales, etc. Este formato apenas es compatible con otros programas debido a los tipos de información adicional que almacena; es por eso que los usuarios de Adobe Photoshop acostumbran a guardar una copia en PSD y otra en otro formato más común como por ejemplo JPEG para poder usarla como imagen final e integrarla en otros archivos.

Los bytes de identificación del formato PSD son [18]:

Posición	Bytes
0	38
1	42
2	50
3	53

Tabla 15 Bytes identificación del formato PSD

Y la información en bytes se almacena en Big-endian de la siguiente forma:

Posición	Tamaño (bytes)	Significado
4	2	Versión
12	2	Número de canales
14	4	Altura (pixels)
18	4	Anchura (pixels)
22	2	Bits por canal
24	2	Modo color

Tabla 16 Estructura del formato PSD

Donde el byte **Mode** puede tener los siguientes valores:

Valor	Modo
0	Monocrono
1	Escala de Grises
2	Paleta de color
3	Color RGB
4	Color CMYK
7	Color Multicanal
8	Halftone*
9	Color Lab

Tabla 17 Modo del formato PSD

*Halftone (método que simula tonos continuos modificando el tamaño y la distancia de puntos)

2.1.2.9. DPX

Es un formato para películas digitales y un estándar ANSI/SMPTE (268M – 2003) derivado del formato de salida del escáner Kodac Cineon que más tarde fue publicado como estándar por SMPTE.

DPX representa la densidad de canal de color de un negativo escaneado en un formato de 10 bits de longitud.

Sus bytes de identificación son [19]:

Posición	Bytes
0	53
1	44
2	50
3	58

Tabla 18 Bytes identificación DPX

Y su estructura de byte en Big Endian es la siguiente:

Posición	**Tamaño (bytes)**	**Significado**
8	8	Versión de bytes_identif
16	4	Tamaño (bytes)
24	4	Tamaño cabecera genérica (bytes)
28	4	Tamaño cabecera industrial (bytes)
32	4	Tamaño definido usuario
36	100	Nombre imagen
136	24	Fecha
160	100	Creador
260	200	Nombre del proyecto
460	200	Derechos
660	4	Clave encriptada

Tabla 19 Estructura de formato DPX

En el software se ha desechado el valor **ditto_key** no considerándose oportuno en nuestro estudio.

El valor **key**, llamado “Clave encriptada” en el software, aparecerá en decimal, siendo el valor -1 cuando no esté encriptado en vez de FFFFFFFF como indica este esquema.

2.1.2.10. JPEG

Se desarrolló a manos del Joint Photographic Expert Group (asociación de fotógrafos profesionales de Estados Unidos) buscando poder almacenar ficheros de imagen con alta calidad pero pudiendo configurar los pesos de dichos ficheros. Utiliza un método de compresión con pérdidas con el mismo nombre que el formato [20].

Dentro del formato JPEG en sí, vamos a diferenciar dos clases distintas denominadas EXIF y JFIF.

Sabremos con cuál de las dos estamos trabajando por el cuarto byte de identificación y gracias a los bytes con posiciones de 6 a 11 que nos los especifica en ASCII.

El formato EXIF se creó para poder almacenar el estado de las cámaras digitales (abertura, disparo, balance de blancos, etc) en las áreas APPn a la hora de guardar la imagen.

El formato JFIF se utiliza en la edición de imágenes profesionales guardando en las áreas APPn datos del copyright, leyendas, gestión de color, etc. Es el más utilizado de los dos.

Sus bytes de identificación en hexadecimal son:

	JFIF	EXIF
Posición	Bytes	Bytes
0	FF	FF
1	D8	D8
2	FF	FF
3	E0	E1

Tabla 20 Bytes identificación JPEG

Y la estructura de la versión JFIF es la siguiente:

Posición	Tamaño (bytes)	Significado
6	5	Tipo
13	1	Unidades de x/y
14	2	Densidad pixel horizontal
16	2	Densidad pixel vertical
18	1	Pixels horizontales en miniatura
19	1	Pixels verticales en miniatura

Tabla 21 Estructura del formato JPEG

Siendo la misma para la versión EXIF a diferencia del byte de posición 3 que será **0xE1** como se ha visto en la tabla anterior.

De estos datos, el programa ignorará los datos de las posiciones 4, 11 y 12 ya que no se consideran importantes en este estudio.

2.2. Formatos de audio

2.2.1. Generalidades

Los formatos de audio se dividen en tres tipos distintos dependiendo del método de compresión que utilicen:

2.2.1.1. Formatos de audio sin comprimir

Se suelen utilizar para el almacenamiento de grabaciones originales. La ventaja de estos formatos es su alta fidelidad pero en su contra tienen el gran espacio que ocupan.

2.2.1.2. Compresión sin pérdidas (Lossless)

La compresión sin pérdidas mantiene la calidad del archivo casi intacta, comprimiendo levemente el archivo para reducir el tamaño en disco. Un método es codificar tanto el silencio como el audio con el mismo número de bits por segundo, otro ejemplo es utilizar tablas de probabilidad como el método Huffman que se basa en crear códigos prefijo asignando códigos con menos bits a los datos con mayor probabilidad de aparición y con mayor número de bits a los que tengan menor probabilidad [21].

2.2.1.3. Compresión con pérdidas

Este método elimina datos que el oído humano es incapaz de percibir. Realmente la calidad del formato apenas se resiente con este método ya que el humano no lo nota pero sí se elimina gran parte de información y por tanto, mucho peso del fichero.

2.2.2. Tipos de archivo

2.2.2.1. MP3

Es un formato de audio digital que utiliza un algoritmo de compresión con pérdidas para disminuir el tamaño del archivo. Dicho algoritmo se basa en principios psicoacústicos que determinan que el ser humano es incapaz de apreciar algunos sonidos, que por tanto, se eliminan [22].

La calidad del sonido se mide en kilobits por segundo. Lo más usado para el formato mp3 es 128kbits/s.

El formato mp3 pertenece a la estandarización del MPEG del que se hablará más adelante.

Este formato, tiene la particularidad de contener una cadena de información adicional como títulos o autores de la pista que se coloca al principio del archivo identificado por los caracteres **ID3.** Esta cadena de bytes indica, además, su longitud, que será imprescindible conocer para el estudio del formato.

Los bytes de identificación del formato son:

Posición	Bytes
0	49
1	44
2	33

Tabla 22 Bytes de identificación de formato MP3

Una peculiaridad del formato mp3 es que la información viene ordenada en bits en vez de en bytes, con lo cual tendremos que crear un algoritmo especial para poder obtener dichos datos.

La estructura del formato después de la cadena de información adicional es de la forma:

POSICION (bits)	TAMAÑO (bits)	SIGNIFICADO
0	11	Bits identificación
11	2	Versión Audio 01 Reservado 10:MPEG v2 11:MPEG v1
13	2	Capa: 00- Reservado 01-Layer III 10- Layer II 11- Layer I
15	1	Protección 0- Si 1- No
16	4	Bitrate(ver tabla)
20	2	Sampling Rate (frecuencia de muestreo) (ver tabla)
22	1	Bit padding
23	1	Bit privado
24	2	Modo canal 00- Stereo

		01- Joint Stereo 10- Canal dual 11- Canal mono
26	2	Modo extensión (ver tabla)
28	1	Copyright
29	1	Original
30	2	Énfasis: 00-Nada 01- 50/15 ms 10- Reservado 11- CCIT J.17

Tabla 23 Estructura formato MP3

Bitrate Index	MPEG 1			MPEG 2	
	Layer I	Layer II	Layer III	Layer I	Layer II
0000	free				
0001	32	32	32	32	8
0010	64	48	40	48	16
0011	96	56	48	56	24
0100	128	64	56	64	32
0101	160	80	64	80	40
0110	192	96	80	96	48
0111	224	112	96	112	56
1000	256	128	112	128	64
1001	288	160	128	144	80
1010	320	192	160	160	96
1011	352	224	192	176	112
1100	384	256	224	192	128
1101	416	320	256	224	144
1110	448	384	320	256	160
1111	Reservado				

Tabla 24 Bitrate para MP3 en Kbps

Velocidad de bit	MPEG 1	MPEG 2
"00"	44100 Hz	22050 Hz
"01"	48000 Hz	24000 Hz
"10"	32000 Hz	16000 Hz
"11"	Reservado	

Tabla 25 Frecuencias de muestreo de MP3

Además de estos datos, y sólo para interés del lector, se añadirá que los ficheros mp3 contienen mucha información fácil de localizar en la cadena de información adicional que será de gran utilidad para el usuario. Dicha información viene dada por etiquetas que se pueden localizar en los datos escritos en ASCII del programa. Con dichas etiquetas podremos identificar desde la fecha de grabación al cantante solista entre otros muchos datos.

Existen un gran número de etiquetas, pero solo se mostrarán las que se han considerado más interesantes para su estudio:

COMM	Comentarios
ENCR	Método de registro encriptado
LINK	Información vinculada
MCDI	Identificador de CD de música
PCNT	Contador de reproducción
SYLT	Letra/texto sincronizado
SYLC	Códigos de tempo sincronizados
TALB	Título del álbum
TBPM	Pulsaciones por minuto
TCOM	Compositor
TCOP	Mensaje de copyright
TDAT	Fecha (DD,MM)
TEXT	Compositor de letra
TFLT	Tipo de fichero

TIME	Tiempo de la grabación (HH,MM)
TIT1	Descripción del grupo
TIT2	Nombre de la canción
TIT3	Subtítulo/descripción más explicita
TLAN	Lengua
TLEN	Longitud
TOAL	Álbum original
TRCK	Número de pista
TYER	Año
WXXX	Link URL del usuario

Tabla 26 Etiquetas MP3

No todos los ficheros tienen que tener las mismas etiquetas ni contener todas. Las más comunes suelen ser los títulos de canciones y álbum o duración de las pistas.

2.2.2.2. MIDI

El sistema MIDI (Musical Instrumental Digital Interface) es un protocolo de comunicación serie que permite a dispositivos como sintetizadores, ordenadores o controladores, comunicarse entre ellos para generar sonidos.

Este sistema consiste en simplificar datos (nota, duración de la nota, fuerza con la que se toca, etc.) en números, consiguiendo así una rápida transmisión entre dispositivos y una gran ligereza de archivos con melodías complejas.

Los bytes de identificación son:

Posición	Bytes
0	4D
1	54
2	68
3	64

Tabla 27 Bytes de identificación de formato MIDI

Y la estructura de los bytes de información es [23]:

Posición	Tamaño	Tipo	Descripción	Valor
0	4	Char[4]	Chunk ID	“MThd” (0x4D546864)
4	4	dword	Chunk size	6 (0x0000006)
8	2	Word	Format type	0 - 2
10	2	Word	Number of tracks	1 – 65,535
12	2	Word	Time division	

Tabla 28 Estructura del formato MIDI

2.2.2.3. WAV

Desarrollado por IBM y Microsoft almacena sonidos tanto mono como estéreo con distintas revoluciones y velocidades de muestreo y la mayoría de veces sin compresión.

Igual que el formato avi de vídeo, wav es un una variante del formato RIFF que almacena la información en paquetes.

Como no suele comprimirse, es un formato ideal para el uso profesional. Si se grabase el sonido con 16bits y una frecuencia de 44100Hz, conseguiríamos una calidad de CD de audio, con la desventaja de que no podemos ocupar más de 4Gb (unas 6,6 horas).

Como todas las variantes del formato RIFF, la cabecera será:

Posición	Bytes
0	52
1	49
2	46
3	46

Tabla 29 Bytes de identificación de formato RIFF

Definiéndose si es formato wav en el byte de posición 8.

Posición	Bytes
8	57
9	41
10	56
11	45

Tabla 30 Bytes de identificación de formato WAV

La estructura del formato es de la forma [24]:

Posición	**Tamaño**	**Valor**	**Endian**
8	4	Formato	Big
22	2	Nº Canales	Little
24	4	Frecuencia de muestreo	Little
28	4	ByteRate	Little
34	2	Bits por muestra	Little

Tabla 31 Estructura del formato WAV

2.2.2.4. FLAC

Este formato (Free Lossless Audio Codec) es muy similar al formato mp3 exceptuando que el Flac codifica sin pérdidas [25]. No reduce excesivamente el tamaño del archivo original (aproximadamente un tercio) dado que no elimina información. Gracias a esta calidad, se convirtió en uno de los formatos favoritos para la venta de música por internet aunque actualmente está en desuso.

Los bytes de identificación del formato FLAC son:

Posición	Bytes
0	66
1	4C

2	61
3	43

Tabla 32 Bytes de idenficicación de formato FLAC

El resto de información, vendrá dada en bits en vez de bytes siguiendo la estructura que se describe a continuación.

Tras los bytes de identificación, se encuentra un bloque denominado "METADATA BLOCK" que se subdivide en **metadata block header** y **metadata block data**. A su vez, **metadata block data** se divide en:

- Metadata_block_streaminfo
- Metadata_block_padding
- Metadata_block_application
- Metadata_block_seektable
- Metadata_block_vorbis_comment
- Metadata_block_picture

Este software, atendiendo a su principio de lectura de cabeceras, sólo analizará **metadata block header** y el primer bloque de **metadata block data** (metadata block streaminfo).

Las estructuras de estos dos bloques son las siguientes:

METADATA _BLOCK HEADER	
bits	Significado
<1>	last-metadata-block flag
	0: STREAMINFO
	1: PADDING
	2: APPLICATION
	3: SEEKTABLE
	4: VORBIS_COMMENT
	5: CUESHEET
	6:PICTURE
	7-126: reserved
<7>	127: invalid

<24>	tamaño en bytes del metadato a seguir (no incluye el tamaño del METADATA_BLOCK_HEADER)

Tabla 33 Estructura de Metadata_block Header

METADATA _BLOCK STREAMINFO	
bits	significado
<16>	Tamaño mínimo del bloque (en muestras)
<16>	Tamaño máximo del bloque (en muestras)
<24>	Tamaño mínimo de frame (en bytes)
<24>	Tamaño máximo de frame (en bytes)
<20>	Frecuencia de muestreo en Hz
<3>	Nº de canales (1…8)
<5>	Bits por muestra
<36>	Total de muestras en steam

Tabla 34 Estructura de Metadata_block Streaminfo

2.2.2.5. AIFF

Es un formato de audio que desarrolló Apple en 1988 basándose en un antiguo formato conocido como IFF (Interchange File Format) [26].

En el formato AIFF se comprime sin pérdidas siendo idóneo para usarse a nivel profesional aunque con la desventaja de ocupar mucho espacio: unos 50MB por 5 minutos de audio estéreo a una frecuencia de muestreo de 44.1kHz y 16 bits.

En este tipo de formato los datos vienen organizados en pequeñas partes, conocidas como chunks, que se identifican con cuatro bytes de identificación. Aunque este formato tiene varios tipos de chunks nosotros solo estudiaremos las dos que nos aportan datos interesantes y que además son las únicas obligatorias: **Common Chunk (COMM)** que nos dará información sobre el sonido y **Sound Data Chunk (SSND)** que nos hablará de los sample frames, todo ello en orden Big-endian.

Los bytes de identificación de este formato son:

Posición	Bytes
0	46
1	4F
2	52
3	4D

Tabla 35 Bytes de identificación de formato AIFF

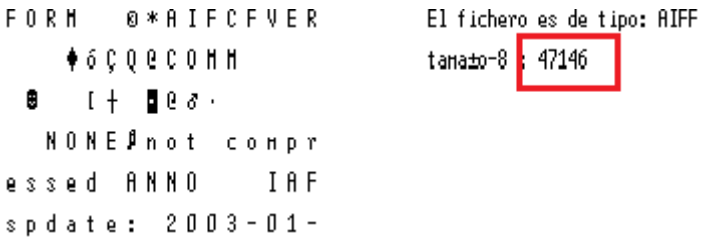
Que son los caracteres ASCII “FORM”.

La estructura de la cabecera es:

```
Offset    Length    Contents
  0       4 bytes   "FORM"
  4       4 bytes   <File size - 8>
  8       4 bytes   "AIFF"
```

Ilustración 3 Estructura de formato AIFF

Habrá que tener en cuenta a la hora de leer el fichero que al valor representado por tamaño hay que sumarle los 8 bytes anteriores:



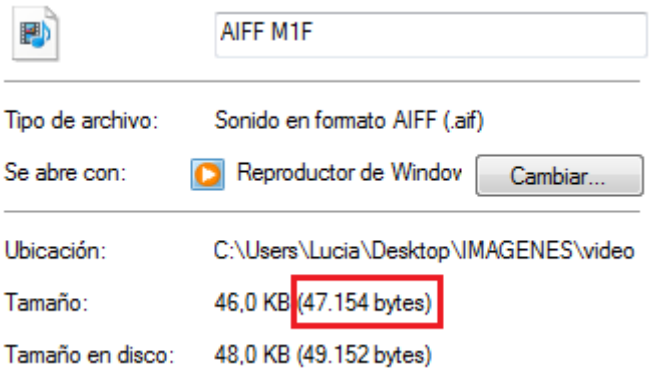


Ilustración 4 Comprobación de tamaño

La del chunk COMM:

```
COMM chunk: Must be defined
  0       4 bytes  "COMM"
  4       4 bytes  <COMM chunk size>  // (=18)
  8       2 bytes  <Number of channels(c)>
 10       4 bytes  <Number of frames(f)>
 14       2 bytes  <bits/samples(b)>  // 1..32
 16      10 bytes  <Sample rate
```

Y la del chunk SSND:

```
SSND chunk: Must be defined
  0       4 bytes  "SSND"
  4       4 bytes  <Chunk size(x)>
  8       4 bytes  <Offset(n)>
 12       4 bytes  <block size>       // (=0)
 16     (n)bytes   Comment
 16+(n) (s)bytes  <Sample data>      // (s) := (x) - (n) - 8
```

Como se ve, las chunks dependen del tamaño de la anterior, por tanto, no se podrán encontrar por orden de bytes sino que se tendrá que recurrir a su análisis visual en el espacio de los caracteres ASCII del programa.

2.2.2.6. WMA

Con WMA nos podemos referir a un códec de audio o a un formato de audio comprimido desarrollado por Microsoft.

Existen cuatro tipos distintos: El primero es el códec original WMA que se creó para competir con el formato MP3, luego apareció el WMA Pro que mejoró el primero e

incorporó la capacidad se soportar audio surround y de alta resolución. El tercer formato es el WMA Lossness que comprime el audio sin pérdida y el cuarto el WMA Voice que se utiliza para contenido hablado con mucha compresión y tasa de bits muy baja [27].

Este formato comparte bytes de identificación y estructura con el formato WMV (se hablará de él más tarde) dado que pertenecen al formato ASF pero tienen la desventaja para este proyecto de contener una infraestructura para proteger el copyright muy severa por tanto, no se podrá analizar con este programa.

Los bytes de identificación son:

Posición	Bytes
0	30
1	26
2	B2
3	75
4	8E
5	66
6	CF
7	11
8	A6
9	D9
10	00
11	AA
12	00
13	62
14	CE
15	6C

Tabla 36 Bytes de identificación de formatos WMA y WMV

2.3. Formatos de vídeo

2.3.1. Generalidades

Los formatos de video que se van a analizar son de tipo contenedor. Los formatos de este tipo se caracterizan porque el archivo contiene varios tipos de datos ya sean audio, vídeo, capítulos, subtítulos, etc. comprimidos con distintos códecs y almacenados conjuntamente. Para reproducir estos archivos, se demultiplexan las pistas (de audio y vídeo) siempre que se conozca la estructura y cada una de ellas se interpreta por el decodificador. Obviamente, nunca se podrá reproducir uno de estos ficheros si el reproductor no tiene los decodificadores necesarios ya que será incapaz de interpretarlos.

2.3.2. Tipos de formatos de vídeo

2.3.2.1. MPEG

El estándar MPEG (Moving Picture Experts Group) fue creado por ISO e IEC para estandarizar el audio y video.

Los algoritmos de MPEG comprimen la información en pequeños paquetes para facilitar la transmisión y la descompresión. La compresión utilizada en este estándar es la conocida Trasformación discreta del coseno (DCT).

Los principales formatos de compresión son:

- **MPEG-1**: Es el estándar inicial de compresión de audio y vídeo. Proporciona vídeo con resolución de 352x240 a 30 frames por segundo (fps) e incluye el formato de compresión de audio capa III (mp3).
- **MPEG-2**: Estándar de audio y vídeo para difusión de calidad de televisión. Aporta resoluciones de 720x480 y de 1280x720 a 60 fps con calidad de CD de audio. Se utiliza para la mayoría de estándares de televisión, hasta HDTV, para TV por sátelite y TV digital por cable entre otros.

- **MPEG-3**: Se diseñó para HDTV pero se abandonó instaurándose éste en MPEG-2.
- **MPEG-4**: Estándar para compresión de gráficos y video basado en MPEG-1, MPEG-2 y Apple QuickTime. Este formato es más pequeño de lo normal por eso es idóneo para la transmisión por un ancho de banda estrecho.
- **MPEG-7**: Está diseñado para ser genérico y no para un uso específico. Es un sistema de herramientas para contenido multimedia.
- **MPEG-21**: Es un estándar que define la descripción del contenido y también de los procesos para acceder, almacenar y proteger el copyright del contenido.

El software se centrará en el análisis de los estándares MPEG-1, MPEG-2.

2.3.2.1.1. MPEG-1

Este estándar se publica en la norma ISO/IEC 11172 que se divide en cinco partes [28]:

- **Parte 1 (11172-1)**: Sistemas de almacenamiento (sincronización de audio, vídeo y demás datos adjuntos)

  Especifica la disposición y los métodos de almacenamiento del audio, vídeo y otros datos codificados. Está diseñado para el almacenamiento en los medios de comunicación y transmisión por canales de datos.

  En esta parte de la norma se definen los formatos "program stream" y "elementary stream" que se especificarán más adelante.

- **Parte 2 (11172-2)**: Vídeo (contenido de vídeo comprimido)

  Basándose en la percepción del ojo humano, este formato reduce o descarta frecuencias y áreas de la imagen que el humano no es capaz de apreciar. Además se explota temporal y espacialmente la redundancia común del vídeo, consiguiendo con estos medios una gran compresión de datos. Algunos de estos métodos son los siguientes:

- Submuestreo: Debido a que el ojo humano es mucho más sensible a la variación de brillo que a la de color, el espacio de color se divide en luminancia (brillo), y crominancia (color) que a su vez se divide en Cr (croma roja) y Cb (croma azul) y se reduce a la mitad la información de color con respecto a la de brillo.

  Los tipos de submuestreo más comunes son:

  - 4:2:2, Dos pixeles de crominancia por cada cuatro de luminancia.
  - 4:2:0, Un pixel de crominancia por cada cuatro de luminancia (1x4 horizontal).
  - 4:1:1, Un pixel de crominancia por cada cuatro de luminancia (2x2).
  - 4:4:4, Misma proporción de luminancia que de crominancia, es decir, no se realiza submuestreo

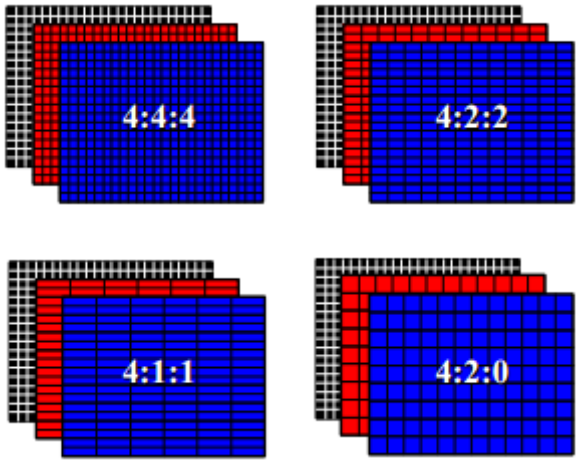


Ilustración 5 Ejemplo de submuestreos

- Frames/ blocks:

  El vídeo MPEG-1 contiene tres tipos de imágenes y cada una de ellas tiene una función distinta. Estas imágenes son las imágenes I, P, B.

  Imagen I (Intra Frame): Son los fotogramas clave. Contienen toda la información de la imagen y sirven de referencia para las otras dos. El espacio que se encuentra entre una imagen I y la siguiente se llama GOP (Group of

pictures) y siempre estará limitado. Cuanto mayor sea el GOP mayor será la compresión de la imagen. Un vídeo MPEG-1 o un cambio de escena siempre tendrá que empezar por una de estas imágenes.

Imagen P (Predicted Frame): Estos fotogramas sólo almacenan la diferencia en la imagen en comparación con una imagen I o P anterior. Esta diferencia se calcula con vectores de movimiento.

Imagen B (Bidirectional Frame): Son muy similares a las imágenes P con la diferencia que las B también se pueden referenciar con imágenes posteriores. Las B se referencian de una imagen I o P pero nunca de una B.

- **Parte 3(11172-3)**: Audio (contenido del audio comprimido)

La parte 3 de esta norma habla sobre la codificación del audio. Al igual que ocurría con el vídeo, este formato basándose en la psicoacústica elimina frecuencias inaudibles para el ser humano o que están enmascaradas por otras.

- **Parte 4 (11172-4)**: Pruebas de conformidad

Cubre las pruebas de conformidad. (No es relevante para el programa)

- **Parte 5 (11172-5)**: Referencia de software

Códigos C de referencia respecto la codificación y decodificación. (no se han utilizado para el programa)

2.3.2.1.2. MPEG-2

El formato MPEG-2 e creó para almacenar televisión digital por cable, satélite o terrestre y como formato de películas que se almacenan en DVD, disco duros, memorias flash, etc.

El MPEG-2 está estandarizado por la norma ISO/IEC 13818 que se divide en once partes [29] aunque sólo serán de interés para el proyecto las tres primeras:

- **Parte 1 (13818-1)**: Sistemas

Igual que en el formato MPEG-1, en este apartado de la norma se describen los formatos contenedores "transport stream" y "elementary stream" que se explicarán en el apartado siguiente.

- **Parte 2 (13818-2):** Vídeo

  Esta parte es muy similar a la del MPEG-1 aunque se añade la mejora de soportar vídeos con entrelazado (formato utilizado por las televisiones).
- **Parte 3 (13818-3):** Audio

  Mejora al estándar MPEG-1 añadiendo la codificación de programas de audio con más de dos canales.
- **Parte 4(13818-4):** Pruebas de conformidad
- **Parte 5 (13818-5):** Software de simulación
- **Parte 6 (13818-6):** Extensiones para DSM-CC
- **Parte 7 (13818-7):** Advanced Audio Coding (AAC)
- **Parte 8 (13818-8):** Vídeo de 10 bits
- **Parte 9 (13818-9):** Extensión de la interfaz en tiempo real para sistemas decodificadores
- **Parte 10 (13818-10):** Extensiones de conformidad para (DSM-CC)
- **Parte 11 (13818-11):** IPMP en sistemas MPEG-2

2.3.2.1.3. Elementary stream

El Elementary stream es un flujos de bits en bruto de MPEG-1 o MPEG-2 de audio o de videos codificados. Los Elementary Stream sólo pueden contener un tipo de dato, por ejemplo, sólo audio o sólo vídeo. Este tipo de archivo se puede distribuir también por su cuenta.

La estructura de la cabecera de vídeo es la siguiente:

Bytes de identificación:

Posición	Bytes
0	00
1	00
2	01
3	B3

Tabla 37 Bytes de identificación de Elementary Stream

Estructura:

Tamaño (bits)	**Significado**
12	Tamaño horizontal
12	Tamaño vertical
4	Relación de aspecto
4	Frame rate
18	Bitrate
1	Bit marker
10	Tamaño del buffer VBV (tamaño verificado = 16*1024*tamaño del buffer VBV)
1	Constrained parameters flag
1	Carga matriz intra de cuantificación
0 ó 64*8	Intra cuantificador de matriz
1	No carga matriz intra de cuantificación
0 ó 64*8	No intra cuantificador de matriz

Tabla 38 Estructura Elementary stream

El software sólo estudiará hasta el Bitrate ignorando los siguientes datos por no ser de interés para este programa.

La estructura del Elementary Stream de audio es la misma que se ha estudiado anteriormente para el formato mp3, con la diferencia de los bytes de identificación y que en este caso, no existe cadena de información adicional, siendo de la forma:

Posición	Bytes
0	F
1	F
2	F

Tabla 39 Bytes de identificación Elementary stream audio

Estructura:

Tamaño (bits)	**Significado**
1	ID '1'=mpeg1 '0'=mpeg2
2	Layer '11'=1 '10'=2 '01'=3
1	Protección '0'= Protegido '1'= No protegido
4	Bitrate
2	Frecuencia de muestreo
1	Padding
1	Privado
2	Modo '00'=Stereo '01'=joint stereo '10'= dual channel '11'= single channel
2	Modo extensión
1	Copyright

	0= No 1= Si
1	Original 0=No 1=Si
2	Énfasis

Tabla 40 Estructura de Elementary stream audio

2.3.2.1.4. Program Stream

La estructura de este tipo de formato es la siguiente:

Bytes de identificación:

Posición	Bytes
0	00
1	00
2	01
3	BA

Tabla 41 Bytes identificación de Program Stream

Estructura:

Posición	**Tamaño (bits)**	**Significado**
5	2	01b
7	3	Referencia reloj del sistema
10	1	1 bit siempre establecido
11	15	Reloj
26	1	1 bit siempre establecido
27	15	Reloj
42	1	1 bit siempre establecido
43	9	Extensión SCR

52	1	1 bit siempre establecido
53	22	Bitrate (50bytes/s)
72	2	11 bits siempre establecidos
74	5	Reservados
79	3	Longitud stuffing

Tabla 42 Estructura de Program stream

2.3.2.2. AVI

Este formato fue creado por Microsoft en 1992 aunque más adelante fue mejorado.

AVI es la versión de vídeo del RIFF que se ha comentado anteriormente. Es un contenedor que almacena tanto datos de audio como de vídeo en diferentes formatos siendo necesario códecs para la lectura de dichos datos [30].

Para que los datos de audio y de vídeo se puedan reproducir simultáneamente, los datos se almacenan entrelazados para que se sincronicen bien. Además, este formato soporta varios flujos de audio, pudiendo almacenar, por ejemplo, varios idiomas a elegir en una misma película.

Los archivos AVI se dividen en fragmentos llamados chunks. Cada chunk llevará un identificador llamado etiqueta FourCC. El primer fragmento será la cabecera que contendrá la información que interesa al software.

Los bytes de identificación de la cabecera serán los correspondientes a "RIFF" como se explicó anteriormente.

Posición	Bytes
0	52
1	49
2	46

3	46

Tabla 43 Bytes identificación formato RIFF

En el byte de posición 8 se especificará si es tipo AVI con los siguientes bytes:

Posición	Bytes
8	41
9	56
10	49
11	20

Tabla 44 Bytes de indentificación de formato AVI

La cabecera de estos formatos es de la forma que se muestra a continuación, comenzando por el byte de posición 28.

Posición	**Tamaño (bytes)**	**Significado**
28	4	Duración frame (ms)
32	4	Max tasa de bytes por segundo
36	4	Padding granularity
44	4	Total frames
48	4	Frame inicial
52	4	Número de streams
56	4	Tamaño del buffer
60	4	Anchura del video stream
64	4	Altura del video stream

Tabla 45 Estructura idenficiación de AVI

2.3.2.3. WMV

Como hemos citado anteriormente, el formato WMV es la versión para vídeo del formato WMA. Sus bytes de identificación y sus particularidades se han descrito en el apartado de WMA.

2.3.2.4. OGG

Es un formato contenedor que encapsula e interpola datos de audio y de vídeo. Por eso, el formato OGG almacena un número de códecs independientes de vídeo y audio en código abierto.

Está libre de patentes y ha sido incluido en muchos reproductores de audio y vídeo y además está orientado a ser un contenedor a stream (que puede ser leído y escrito en un solo paso) siendo así perfecto para el streaming en internet [31].

Sus bytes de identificación son los siguientes:

Posición	Bytes
0	4F
1	67
2	67
3	53

Tabla 46 Bytes de indentificación de formato OGG

2.3.2.5. FLV

Flash vídeo es un tipo de formato contenedor que se utiliza para la transmisión de vídeo por internet como por ejemplo Youtube.

Para poder reproducirse en la red, es necesario tener instalado el plugin Adobe Flash Player [32].

Sus bytes de identificación son:

Posición	Bytes
0	46
1	4C
2	56

Tabla 47 Bytes de identificación de formato FLV

Y la estructura de la cabecera:

Significado	**Tipo**
Signature	Byte [3]
Version	uint8
Flag	uint8 bitmask
Header Size	uint32_be

Tabla 48 Estructura de formato FLV

Donde **Signature** son los bytes de identificación, **Version** determinará la versión del formato que a día de hoy será 1.

El valor **Flag** puede ser:

- 1 si es un fichero de vídeo
- 4 si es de audio
- 5 si es audio+vídeo

Header Size es el tamaño de la cabecera que para esta versión siempre serán 9 bytes.

3. DESARROLLO DEL PROGRAMA

3.1. Características generales y resultados esperables

El objetivo del programa desarrollado es facilitar el análisis de archivos de imagen, audio y video. El programa EXTRACTOR DE DATOS_LBS identifica el tipo de archivo analizado y, en función de su naturaleza, informa de los parámetros de interés contenidos en las cabeceras de los archivos como se indica a continuación.

Para archivos de imagen, EXTRACTOR DE DATOS_LBS puede llegar a mostrar, por lo general, los siguientes datos:

- Tamaño de la cabecera
- Anchura
- Altura
- Planos de color
- Bits por pixel
- Compresión
- Resolución horizontal
- Resolución vertical

Para archivos de audio, EXTRACTOR DE DATOS_LBS puede llegar a mostrar, por lo general, los siguientes datos:

- Frecuencia de muestreo
- Bitrate
- Número de canales
- Bits por muestra

Para archivos de video, EXTRACTOR DE DATOS_LBS puede llegar a mostrar, por lo general, los siguientes datos:

- Tipos de audio/imagen

Obviamente, en función del formato de imagen, audio o video que estemos considerando, la información contenida en la cabecera variará. Por ello, el listado de propiedades enumerado para cada tipo de archivo debe considerarse como una aproximación general. De este modo, el listado de parámetros enumerados no tienen porqué estar necesariamente contenidos en la cabecera del archivo bajo estudio, de forma que, lógicamente, EXTRACTOR DE DATOS_LBS no será capaz de obtener información alguna de ciertos parámetros cuando los valores de dichos parámetros para el archivo en cuestión no estén contenidos en su cabecera.

3.2. Desarrollo

Como ya se ha mencionado, se va a trabajar en Visual Studio 1010, con lenguaje de programación C++ y arquitectura Doc/View.

El software desarrollado abre un fichero y muestra por pantalla todos los bytes en hexadecimal, en código ASCII y los datos encontrados, por tanto lo primero de todo será, en la clase **View**, en la función encargada de dibujar (**OnDraw**) dividir la pantalla en tres rectángulos que muestren dichos valores.

```
// RECTANGULOS
CRect rcDatosAsci, rcDatosHex, rcInformacion;

// RECTANGULO HEX
rcDatosHex.left = 20;//comienzo de rectangulo por la izquierda
rcDatosHex.top = rcRectClient.top; //rectangulo por arriba
rcDatosHex.right = 440; //final derecha
rcDatosHex.bottom = rcRectClient.bottom - 10;   //final abajo
nMaxX = rcDatosHex.Width() / 26;            //escribe 16 bytes

// RECTANGULO DATOS
rcDatosAsci.left = 440;
rcDatosAsci.top = rcRectClient.top;
rcDatosAsci.right = 600;
rcDatosAsci.bottom = rcRectClient.bottom - 10;

//RECTANGULO INFORMACION
rcInformacion.left = 650;
rcInformacion.top = rcRectClient.top;
rcInformacion.right = rcRectClient.right;
rcInformacion.bottom = rcRectClient.bottom - 10;
```

Ilustración 6 Código de división de pantalla

Donde las variables dignas de destacar son:

Tipo	Nombre	Función
CRect	rcDatosAsci	Rectangulo para información ASCII
CRect	rcDatosHex	Rectángulo para información hexadecimal
CRect	rcInformacion	Rectángulo para información obtenida

Tabla 49 Variables para división de pantalla

Para facilitar la lectura al usuario, se mostrarán 16 bytes por línea.

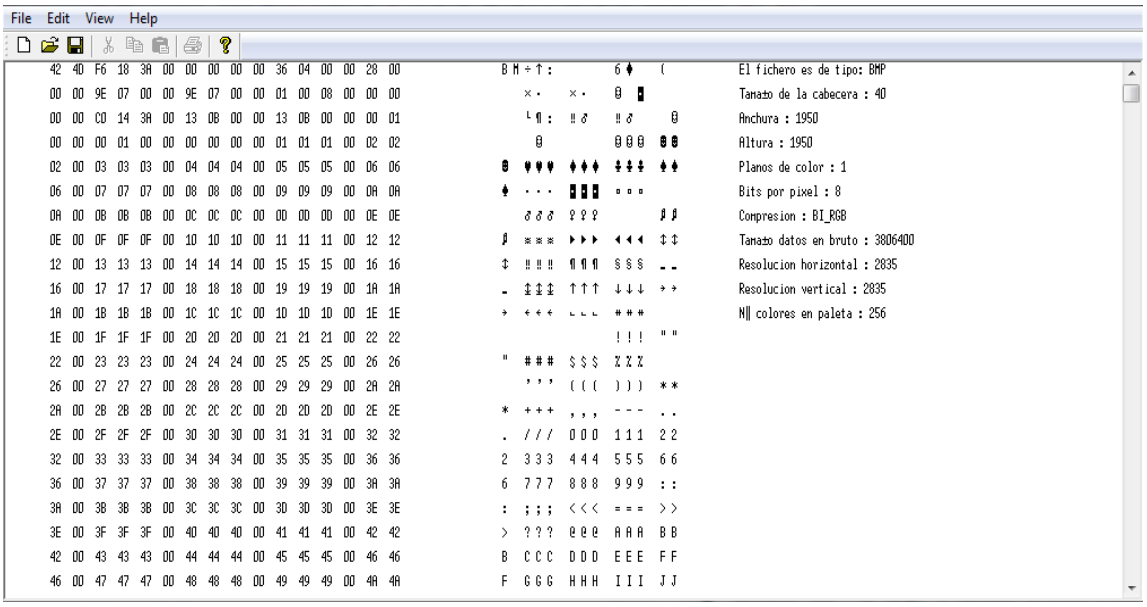
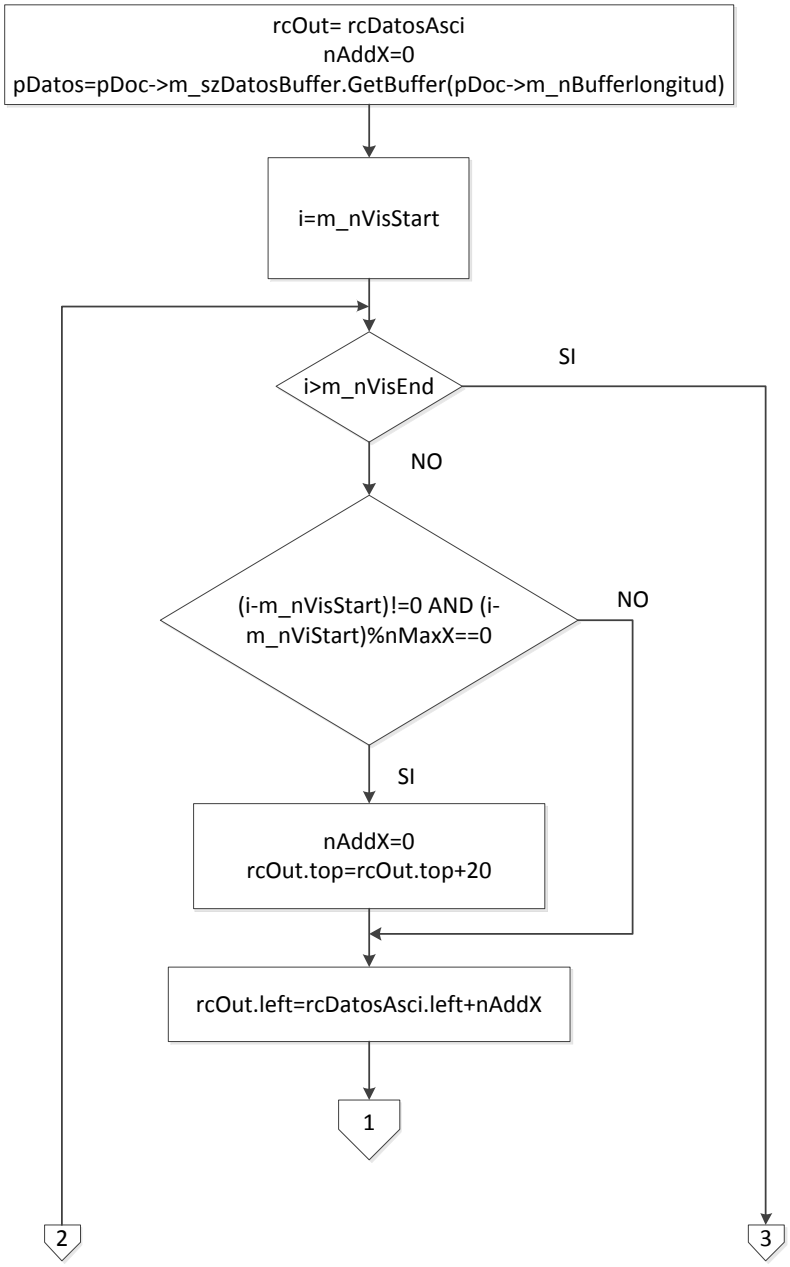
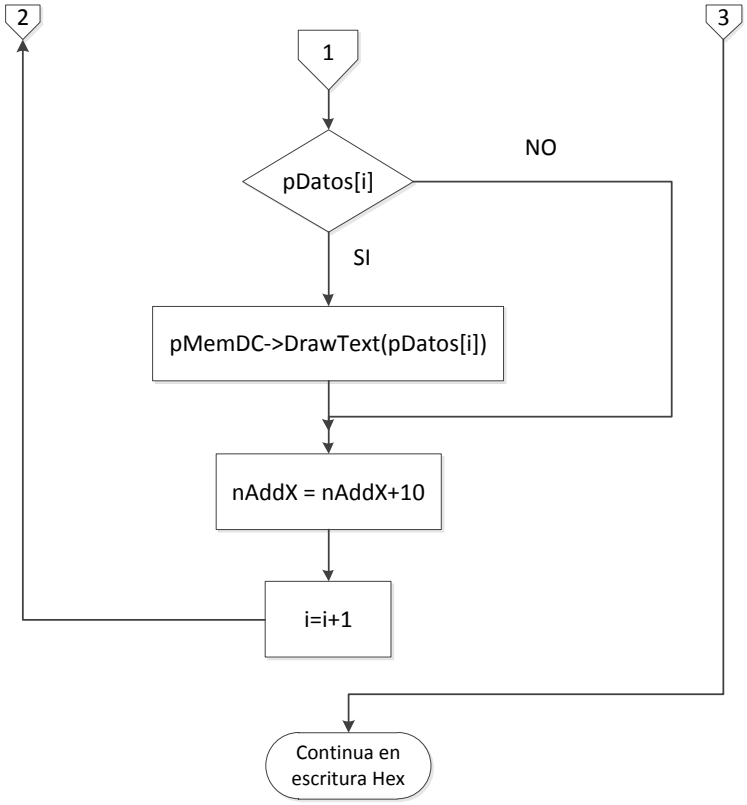


Ilustración 7 Ejemplo del resultado visual

Con los rectángulos ya determinados (rcDatosAsci, rcDatos y rcInfo) se implementará el código para imprimir los tres tipos de datos ya comentados. Para que se entienda correctamente el sistema de impresión, se presenta a continuación un diagrama de flujo para cada uno de los casos.

3.2.1. Diagrama de flujo escritura en ASCII



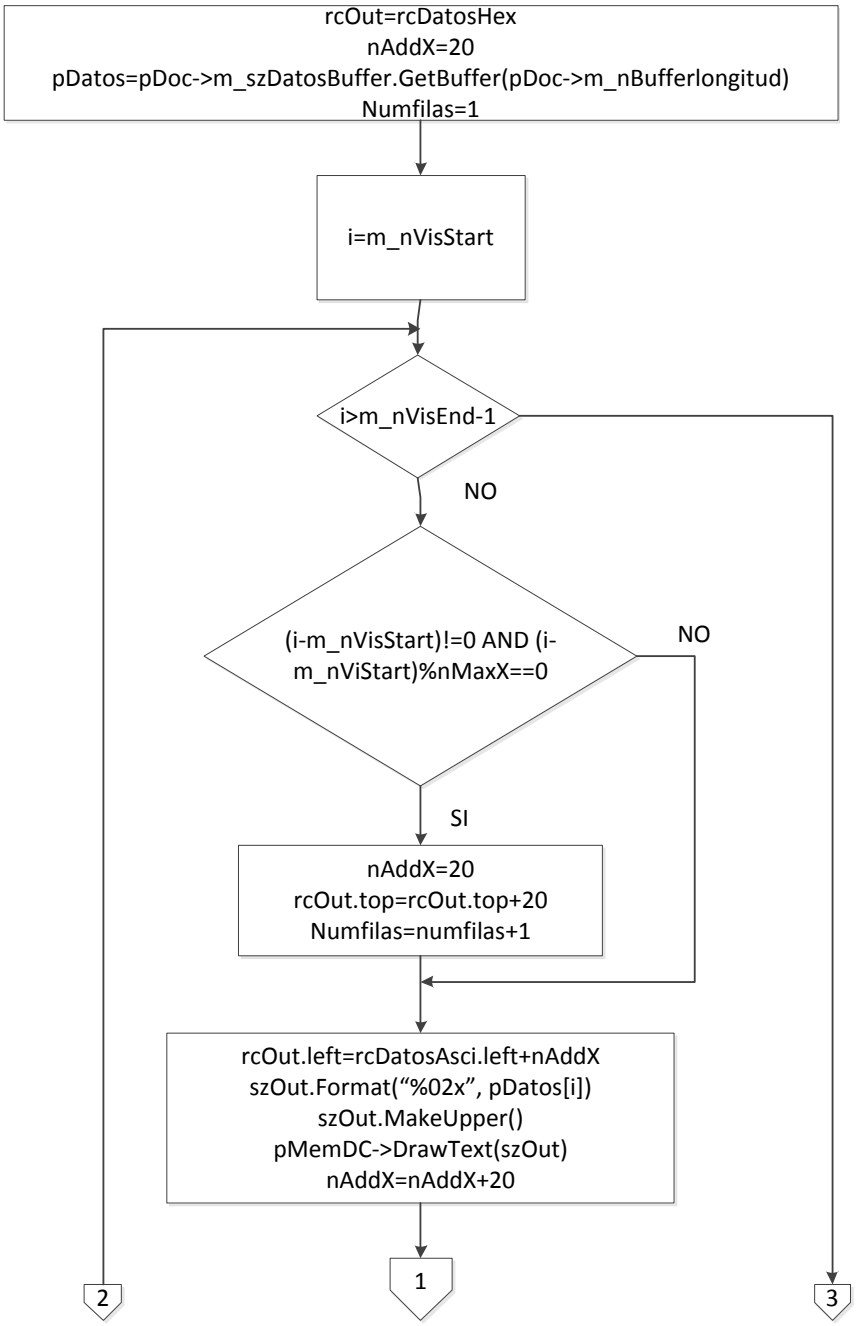


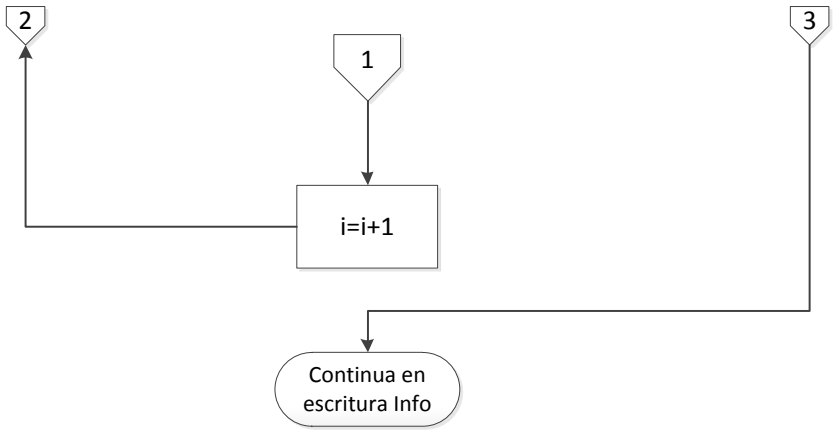
Tipo	Nombre	Función
CRect	rcOut	Rectángulo de salida
LPSTR	pDatos	Puntero al array que guarda los bytes del archivo
int	i	Contador para recorrer area visionado
int	m_nVisStart	Inicio del area del visionado
int	nAddX	Indicador horizontal de columna
CDC*	pMemDC	Variable que guarda los datos de contexto de la pantalla
int	m_nVisEnd	Final del area del visionado

int	nMaxX	Columna final
CextractorDatos_lbsDoc*	pDoc	Puntero al documento

Tabla 50 Variables para la escritura ASCII

3.2.2. Diagrama de flujo de escritura en Hexadecimal

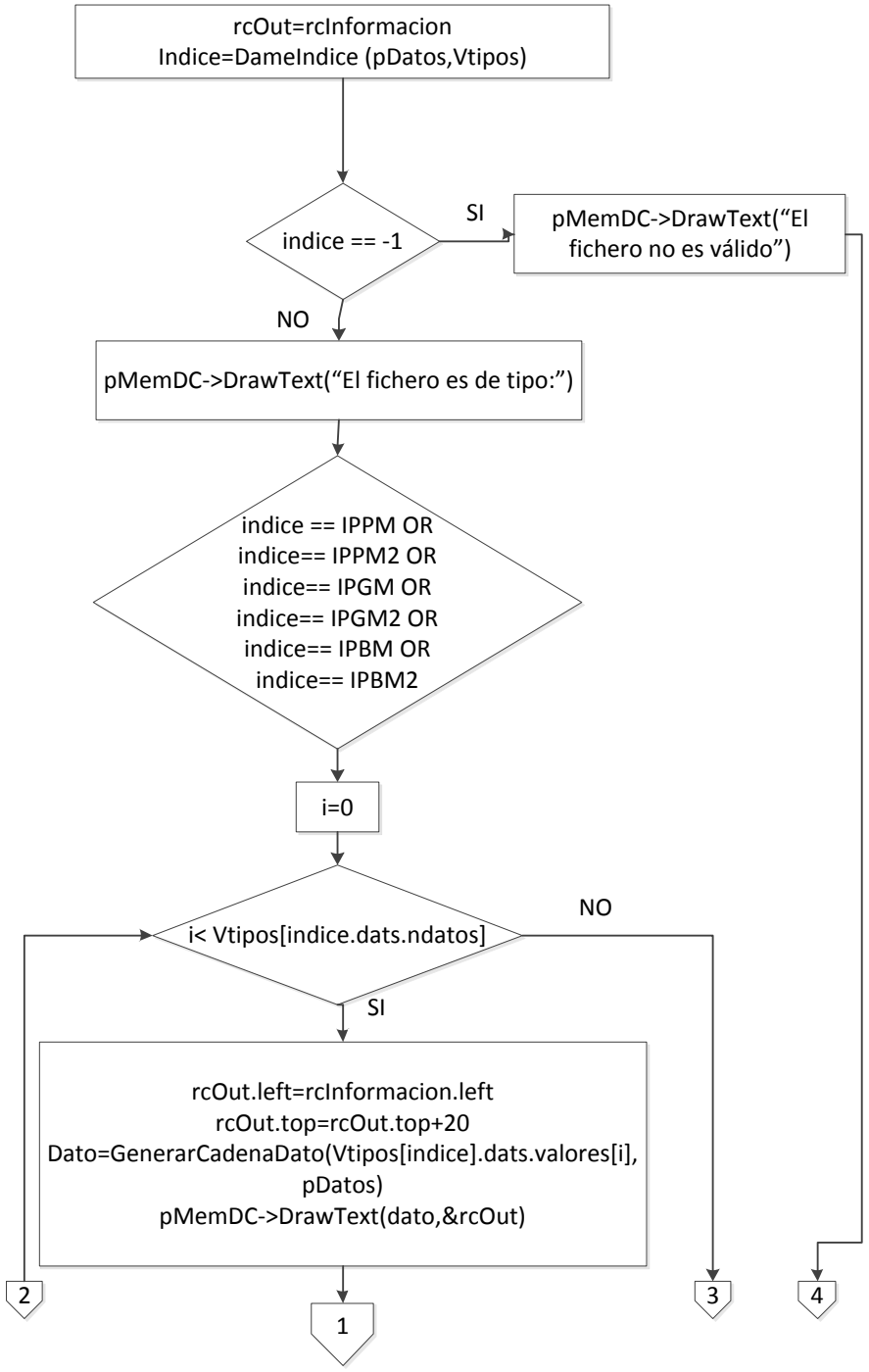


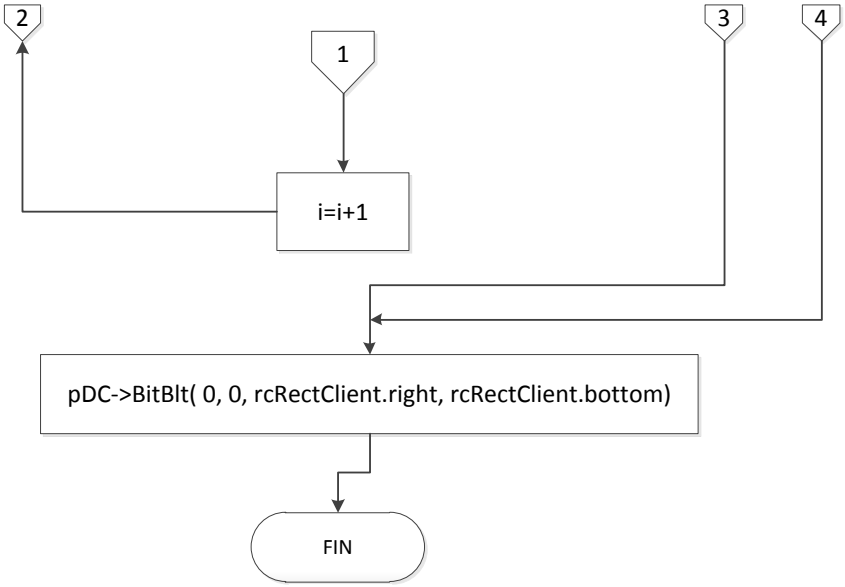


Tipo	Nombre	Función
Crect	rcOut	Rectángulo de salida
LPSTR	pDatos	Almacena la cadena de bytes
int	nAddX	Indicador horizontal de columna
CextractorDatos_lbsDoc *	pDoc	Puntero al documento
int	Numfilas	contador de filas
int	i	contador for
int	m_nVisStart	Inicio del area del visionado
int	m_nVisEnd	Final del area del visionado
int	nMaxX	Columna final
Cstring	szOut	Cadena de salida formateada para mostrarse
CDC*	pMemDC	Variable que guarda los datos de contexto de la pantalla

Tabla 51 Variables para escritura hexadecimal

3.2.3. Diagrama de flujo de escritura de información obtenida





Tipo	Nombre	Función
CRect	rcOut	Rectángulo de salida
int	indice	indica el formato
CDC*	pMemDC	Variable que guarda los datos de contexto de la pantalla
CextractorDatos_lbsDoc*	pDoc	Puntero al documento
int	i	contador for
CString	Dato	Cadena de dato para escribirla
LPSTR	pDatos	Puntero al array que guarda los bytes del archivo
CDC*	pDC	Puntero a CDC

Tabla 52 Variables escritura de información

Con las opciones de visualización ya implementadas, se comenzará con el tratamiento de los ficheros.

Lo primero que hará el programa tras abrir un archivo cualquiera será, en la clase **DOC**, en la función **serialize**, guardarlo en un puntero llamado **pDoc**. Este valor se

guardará en valores decimales, no hexadecimales como necesitaríamos. Por esto y para mayor comodidad, la identificación del formato del fichero se hará en decimal ya que es la parte más importante del software y así evitamos posibles problemas de conversión de bytes de decimal a hexadecimal.

```
void CHexEditorDoc::Serialize(CArchive& ar)
{
    if(ar.IsLoading())
    {
        int tamano;
        tamano = ar.GetFile()->GetLength();
        LPTSTR pszBuffer = m_szDatosBuffer.GetBuffer(tamano);
        m_nBufferlongitud = ar.Read( pszBuffer, tamano); //numero de bytes del archivo

    }
}
```

Ilustración 8 Función Serialize

Ya con el fichero almacenado se comenzará con la identificación del formato, analizando las cabeceras de los archivos.

Para ello, se ha creado la función Cargartipos dónde se ha almacenado toda la información referente a los formatos en varias estructuras del tipo que se muestra en la figura.

```
struct bytes_identif{ //informacion de una cabecera
    int tam;    //tamaño
    int *datos; //bytes de la bytes_identif
    CString nombre;
};
struct dato {   //informacion de un dato de una extension en particular p.ej donde está la duracion
    char* nombre;
    short tipo_dato;
    int byte, tamano;
};
struct datos{   //datos que guardamos de cada tipo de archivo
    int ndatos; //numero de datos (colores, tamaño...)
    struct dato *valores;   //
};
struct tipo{    //informacion para extension
    struct bytes_identif cab;
    struct datos dats;
};
```

Ilustración 9 Estructuras para los formatos

Donde los valores nos especifican:

bytes_identif :

tam: Número de bytes que tiene la cabecera.

***datos**: Almacena los bytes de identificación en un array.

nombre: Nombre del formato.

dato:

***nombre**: Nombre del dato, por ejemplo "Anchura".

tipo_dato: Especifica si el dato está expresado en números, texto, little endian, big endian o es una equivalencia numérica. Es un dato muy importante a la hora de la lectura.

byte: En qué posición comienza el dato.

tamano: Longitud de dicho dato.

datos:

ndatos: Número de datos que vamos a almacenar.

***valores**: Almacena todos los valores de la estructura anterior.

tipo:

cab: Almacena los datos de identificación de cada formato.

dats: Almacena el resto de datos de los formatos.

Dado que la función **CargarTipos** es muy larga y repetitiva, sólo se mostrará un ejemplo de la función donde se han almacenado todas las propiedades del formato PGM:

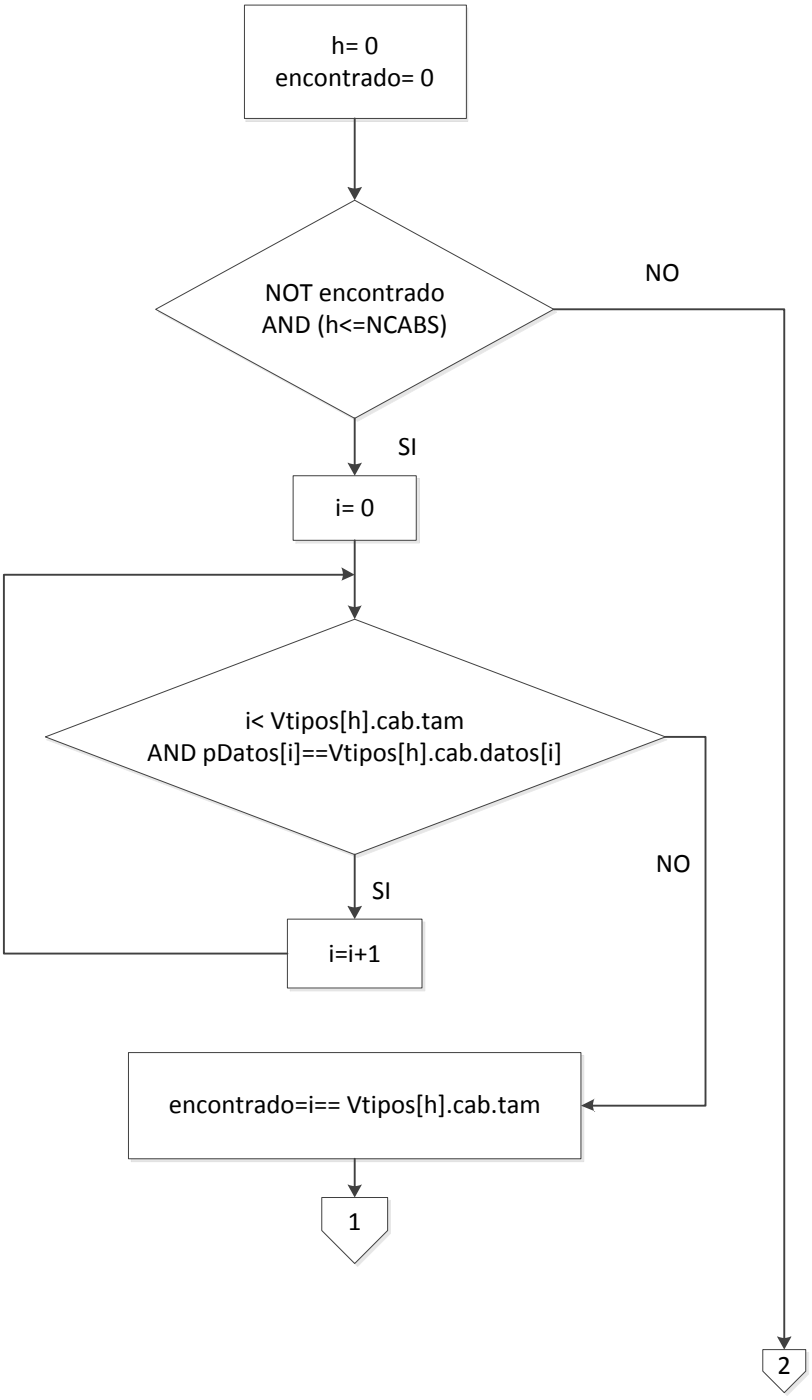
```
////////////PGM////////////////
Vtipos[8].cab.tam=2;
Vtipos[8].cab.nombre="PGM";
Vtipos[8].cab.datos=(int *)malloc (2*sizeof(int));
Vtipos[8].cab.datos[0]=80;
Vtipos[8].cab.datos[1]=50;

Vtipos[8].dats.ndatos=3;
Vtipos[8].dats.valores=(struct dato*)malloc(3*sizeof(struct dato));
Vtipos[8].dats.valores[0].nombre="Anchura";
Vtipos[8].dats.valores[0].byte=3;
Vtipos[8].dats.valores[0].tamano=3;
Vtipos[8].dats.valores[0].tipo_dato= TEXTO;
Vtipos[8].dats.valores[1].nombre="Altura";
Vtipos[8].dats.valores[1].byte=7;
Vtipos[8].dats.valores[1].tamano=3;
Vtipos[8].dats.valores[1].tipo_dato= TEXTO;
Vtipos[8].dats.valores[2].nombre="maximo de colores";
Vtipos[8].dats.valores[2].byte=11;
Vtipos[8].dats.valores[2].tamano=3;
Vtipos[8].dats.valores[2].tipo_dato= TEXTO;
```

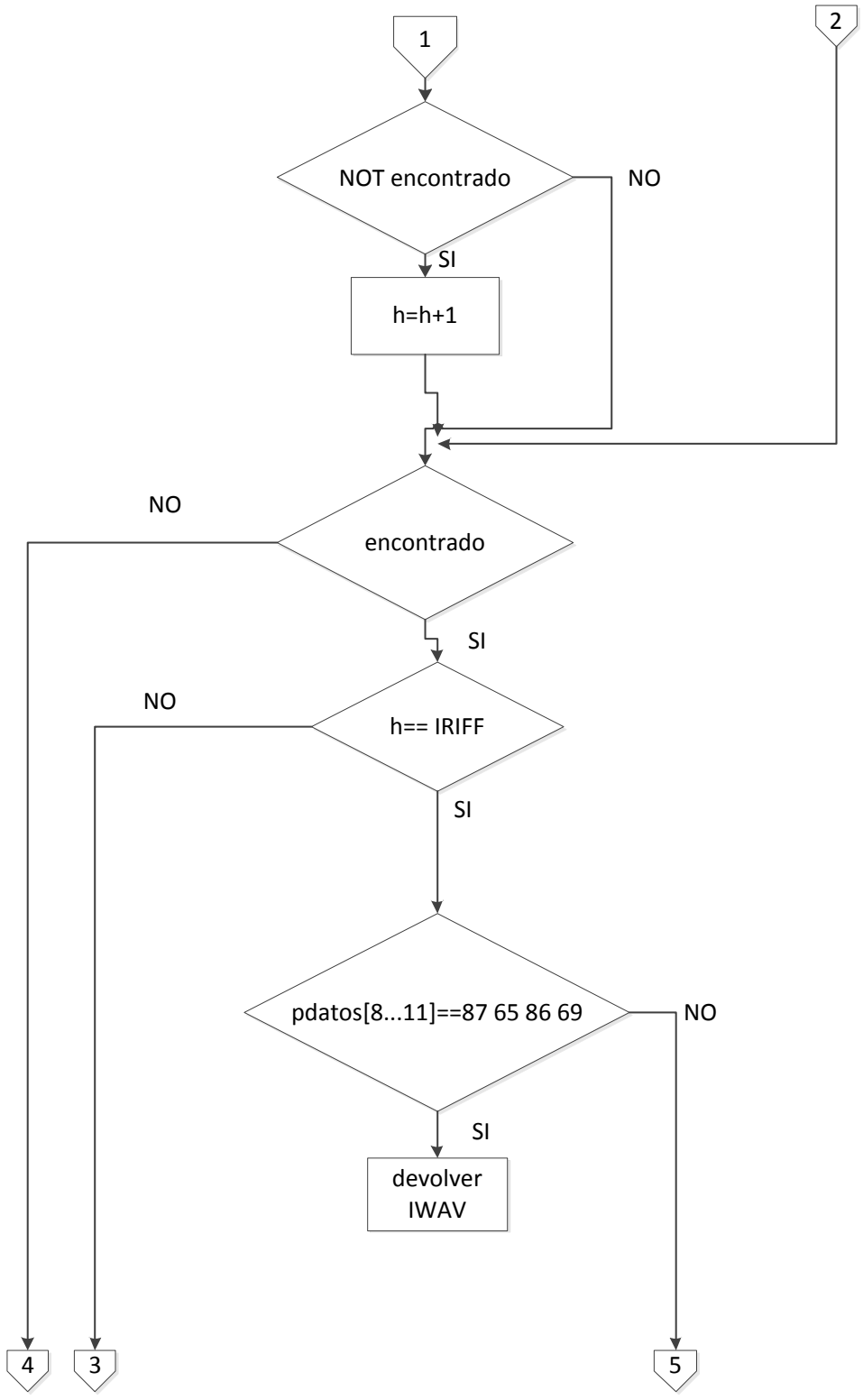
Ilustración 10 Ejemplo de CargarTipos

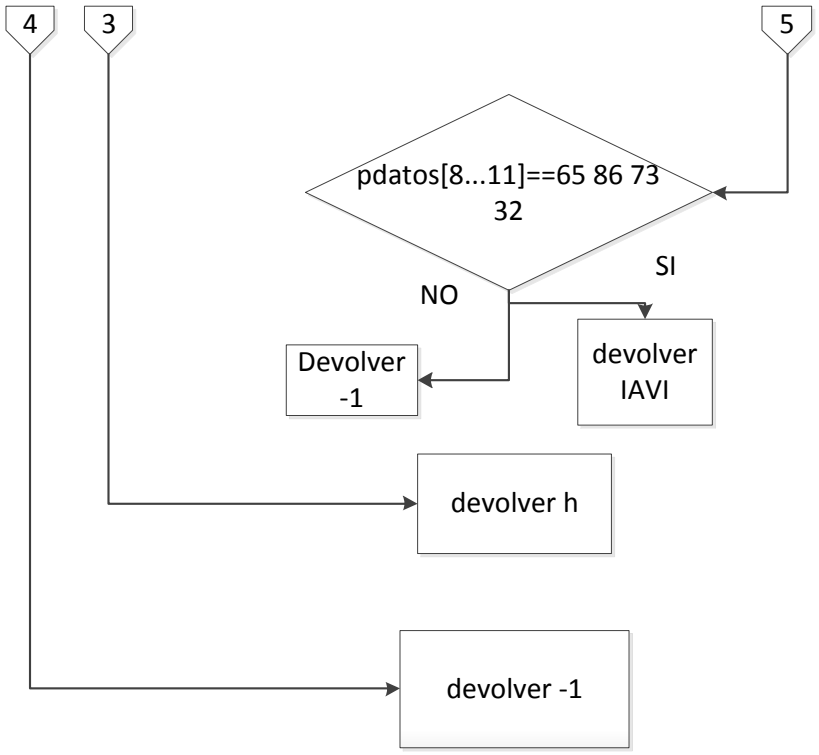
Siguiendo con la estructura del programa, con todos los datos de los diferentes formatos almacenados en la función **CargarTipos** , la función **DameIndice** será la encargada de identificar el tipo de formato de fichero que hemos abierto mediante un sencillo bucle que recorrerá todos los posibles bytes de identificación que tenemos almacenados en la estructura Vtipos hasta dar con el deseado. En caso de que el formato abierto no sea compatible con el software, el programa nos mostrará sus bytes en hexadecimal y ASCII pero nos anunciará que no ha conseguido identificar el formato.

Para explicar con mayor detalle la función **DameIndice**, se muestra a continuación su diagrama de flujo desarrollado paso a paso y el código de la función.

3.2.4. Diagrama de flujo de la función DameIndice




1
2
NOT encontrado
NO
SI
h=h+1
NO
encontrado
SI
NO
h== IRIFF
SI
pdatos[8...11]==87 65 86 69
NO
SI
devolver
IWAV
4
3
5


4
3
5
pdatos[8...11]==65 86 73 32
SI
NO
Devolver -1
devolver IAVI
devolver h
devolver -1

Tipo	Nombre	Función
int	h	contador para Vtipos
BOOL	encontrado	es 1 si encuentra el formato
int	i	contador for
Array struct tipo	Vtipos	Vector de tipos con información de todos los datos de cada tipo
LPSTR	pDatos	Puntero al array que guarda los bytes del archivo

Tabla 53 Variables para la función DameTipos

```
int CHexEditorView::DameIndice(LPSTR pDatos, struct tipo Vtipos[])
{
    int h=0;
    bool encontrado = false;
    while(!encontrado && h<=NCABS)
    {
        int i=0;
        while((i<Vtipos[h].cab.tam)&&((byte)pDatos[i]==Vtipos[h].cab.datos[i]))
            i++;
        encontrado= i==Vtipos[h].cab.tam;
        if (!encontrado)
            h++;
    }
    if (encontrado)
        if (h== IRIFF)
        {
            if (pDatos[8]== 87 && pDatos[9]== 65 && pDatos[10]== 86 &&pDatos[11]== 69) //WAV
                return IWAV;
            else if (pDatos[8]== 65 && pDatos[9]== 86 && pDatos[10]== 73 && pDatos[11]==32) //AVI
                return IAVI;
            else
                return -1;
        }
        else
                return h;
    else
        return -1;
}
```

Ya identificado el formato, se pasará a imprimir por pantalla toda información que se pueda obtener de las cabeceras.

Para mayor comodidad a la hora de imprimir, se ha creado una función específica para leer los datos dependiendo de cómo estén escritos llamada **GenerarCadenaDato**. Probablemente esta función sea la más importante de todo el programa, recomendándose así su estudio detallado.

Para saber cómo están escritos los datos se crea la variable **tipo_dato**, que ya hemos comentado antes, que nos lo especificará con las siguientes constantes:

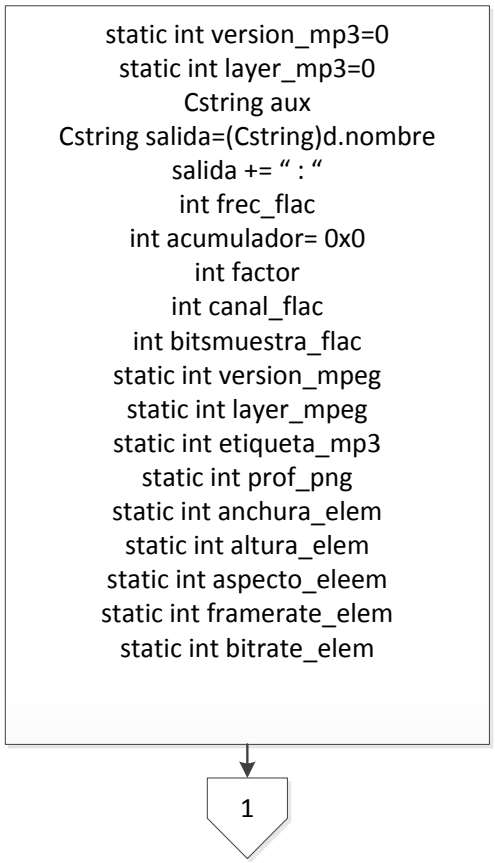
- Cuando se encuentra el valor **TEXTO** se habla de un tipo de dato que ya viene escrito en código ASCII y que simplemente se tiene que imprimir por pantalla sin ningún tipo de conversión.
- El valor **NUMERO_BE** es un dato numérico que viene dado en orden Big-endian donde, como se ha explicado anteriormente, el byte de mayor peso es que se encuentra a la izquierda. Al contrario que el valor **NUMERO_LE** que especifica que es un dato en Little-endian y que su byte de mayor peso es el de la derecha.

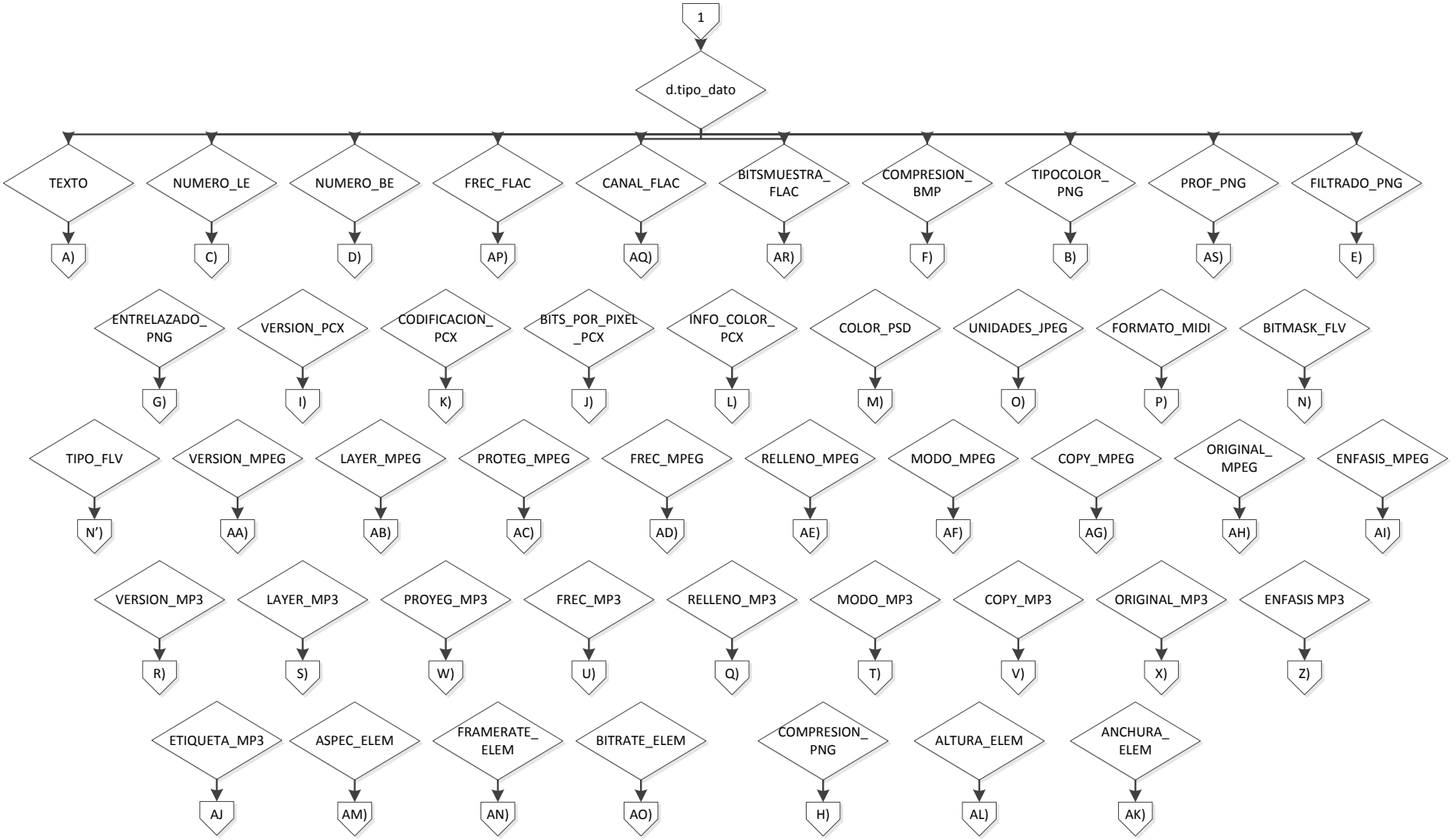
Además de estos tres tipos de escritura, se han definido varias constantes que se utilizar para poder escribir datos representados con equivalencias como por ejemplo cuando encontramos "compresión=1" y realmente significa un tipo de compresión específico.

La función **GenerarCadenaDato** se basa en un switch que dependiendo del valor que tenga **tipo_dato** nos imprimirá el texto de una forma u otra. Dado que el diagrama de flujo de la función **GenerarCadenaDato** es largo y enrevesado, debido al gran número de "switch" que hay, se presentará primero un diagrama global de la función con referencias a posteriores diagramas más detallados que facilitarán la comprensión del algoritmo.

3.2.5. Diagrama de la función GenerarCadenaDato

*No se han dibujado todas las líneas para no complicar más el diagrama. Todos los casos proceden del switch **d.tipo_dato**



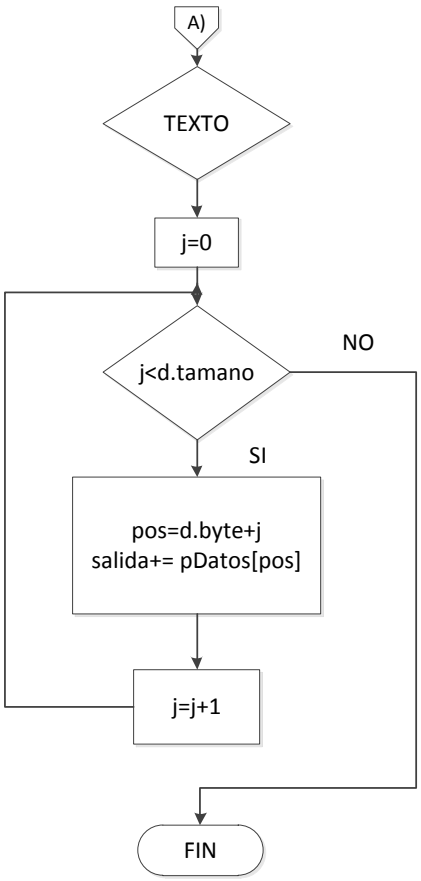

1
d.tipo_dato
TEXTO
NUMERO_LE
NUMERO_BE
FREC_FLAC
CANAL_FLAC
BITSMUESTRA_ FLAC
COMPRESION_ BMP
TIPOCOLOR_ PNG
PROF_PNG
FILTRADO_PNG
A)
C)
D)
AP)
AQ)
AR)
F)
B)
AS)
E)
ENTRELAZADO_ PNG
VERSION_PCX
CODIFICACION_ PCX
BITS_POR_PIXEL _PCX
INFO_COLOR_ PCX
COLOR_PSD
UNIDADES_JPEG
FORMATO_MIDI
BITMASK_FLV
G)
I)
K)
J)
L)
M)
O)
P)
N)
TIPO_FLV
VERSION_MPEG
LAYER_MPEG
PROTEG_MPEG
FREC_MPEG
RELLENO_MPEG
MODO_MPEG
COPY_MPEG
ORIGINAL_ MPEG
ENFASIS_MPEG
N')
AA)
AB)
AC)
AD)
AE)
AF)
AG)
AH)
AI)
VERSION_MP3
LAYER_MP3
PROYEG_MP3
FREC_MP3
RELLENO_MP3
MODO_MP3
COPY_MP3
ORIGINAL_MP3
ENFASIS MP3
R)
S)
W)
U)
Q)
T)
V)
X)
Z)
ETIQUETA_MP3
ASPEC_ELEM
FRAMERATE_ ELEM
BITRATE_ELEM
COMPRESION_ PNG
ALTURA_ELEM
ANCHURA_ ELEM
AJ
AM)
AN)
AO)
H)
AL)
AK)

De esta manera, si el dato es **TEXTO** su algoritmo de impresión será:

```
/**************************/
/*      texto en ascii     */
/**************************/
case TEXTO:
    for(int j=0;j<d.tamano;j++) //por cada byte del dato
    {
        int pos= d.byte+j;
        salida+= pDatos[pos];
    }
    break;
```

Ilustración 11 Código para la escritura de texto

Y el diagrama de flujo:



Tipo	Nombre	Función
int	j	contador para cada byte del dato

Struct dato	d	dato
int	pos	Posición del byte
Array struct tipo	Vtipos	Vector de tipos con información de todos los datos de cada tipo
CString	Salida	Imprime por pantalla el resultado
LPSTR	pDatos	Puntero al array que guarda los bytes del archivo

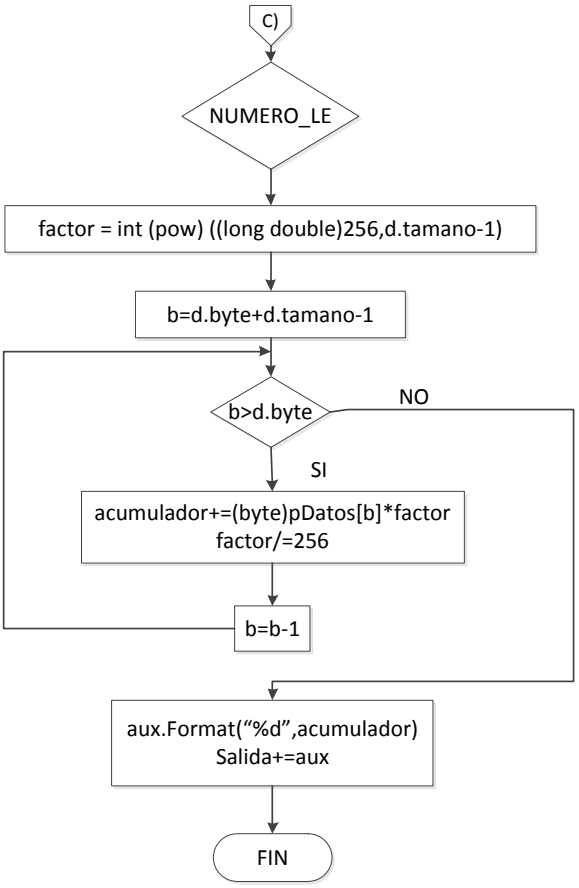
Tabla 54 Variables para la función de escribir texto

Si es **NUMERO_LE** :

```
/*************************/
/*      little endian    */
/*************************/
case NUMERO_LE:
    factor =(int)pow((long double)256,d.tamano-1);
    for(int b=d.byte+d.tamano-1; b>=d.byte; b--)    //acumulamos cada byte con su peso correspondiente
    {
        acumulador+=(byte)pDatos[b]*factor;
        factor/=256;
    }

    aux.Format("%d",acumulador);
    salida+=aux;
    break;
```

Ilustración 12 Código para escritura de Little endian

Y su diagrama de flujo es:



Tipo	Nombre	Función
int	factor	Factor para conversión
int	b	Ultimo byte del dato
Struct dato	d	dato
int	acumulador	Acumula para cálculo del LE
LPSTR	pDatos	Puntero al array que guarda los bytes del archivo
CString	aux	Conversión entero cadena
CString	salida	Imprime por pantalla el resultado

Tabla 55 Variables para la escritura de Little endian

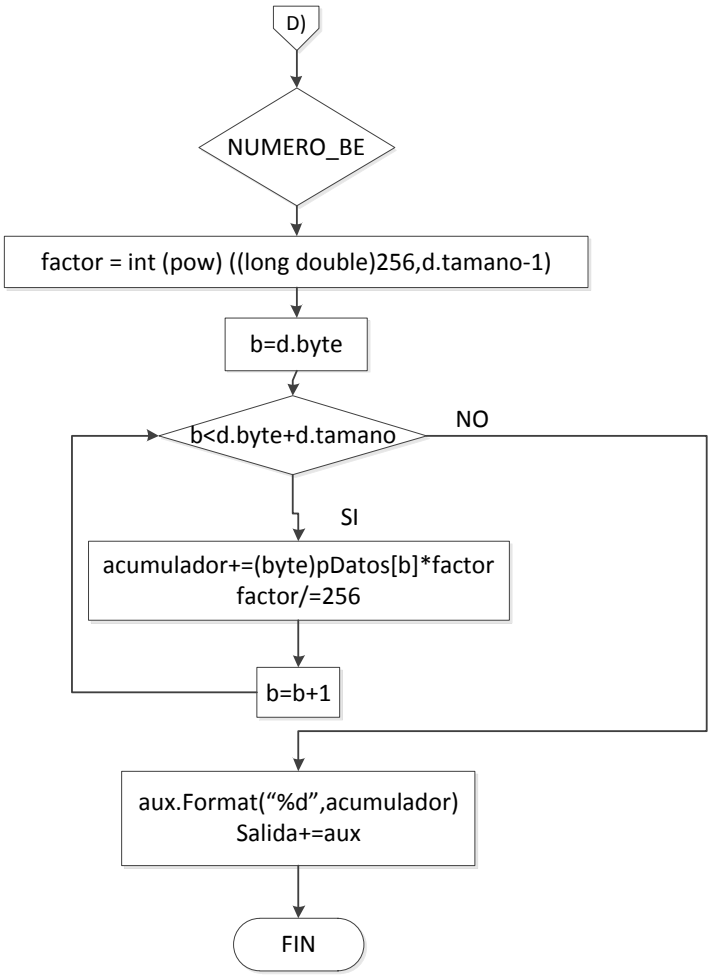
En caso de **NUMERO_BE**:

El código es:

```
/**************************/
/*      big endian        */
/**************************/
case NUMERO_BE:
        factor =(int)pow((long double)256,d.tamano-1);
        for(int b=d.byte; b<d.byte+d.tamano; b++)   //acumulamos cada byte con su peso correspondiente
        {
            acumulador+=(byte)pDatos[b]*factor;
            factor/=256;
        }
    aux.Format("%d",acumulador);
    salida+=aux;
    break;
```

Ilustración 13 Código para escritura de Big endian

Explicándose con mayor claridad con su diagrama de flujo:



Tipo	Nombre	Función
int	factor	Factor para conversión
int	b	Último byte del dato
Struct dato	d	Dato
int	acumulador	Acumula para el cálculo del BE
LPSTR	pDatos	Puntero al array que guarda los bytes del archivo
CString	aux	Conversión entero cadena
CString	salida	Imprime por pantalla el resultado

Ilustración 14 Variables para la función de Big endian

Para el resto de constantes, utilizaremos diferentes métodos de impresión, según requiera el formato, basándonos en cadenas de switch que determinarán el tipo de dato al que equivalen unos valores prefijados.

Dichas constantes tendrán los valores siguientes:

CONSTANTE	VALOR	FUNCIONALIDAD
NCABS	40	Máximo de formatos admitidos
NUM_FRAMES	1	Número de frames
TEXTO	0	Si la información viene dada en ASCII
NUMERO_LE	1	Si la información viene dada en Little endian
NUMERO_BE	2	Si la información viene dada en Big endian
COMPRESION_BMP	3	Tipo de compresion BMP
TIPOCOLOR_PNG	4	Tipo de color PNG
FILTRADO_PNG	5	Tipo de filtrado PNG
COMPRESION_PNG	6	Tipo de compresion PNG
ENTRELAZADO_PNG	7	Entrelazado PNG
VERSION_PCX	8	Versión PCX
CODIFICACION_PCX	9	Codificación PCX
BITS_POR_PIXEL_PCX	10	Bits por pixel PCX
INFO_COLOR_PCX	11	Información del color PCX
COLOR_PSD	12	Tipo color PSD
UNIDADES_JPEG	13	Unidades JPEG
FORMATO_MIDI	14	Formato MIDI
PROF_PNG	15	Profundidad de bit de PNG

BITMASK_FLV	16	Bitmask FLV
TIPO_FLV	18	Tipo FLV
VERSION_MP3	19	Versión MP3
BITRATE_MP3	20	Bitrate MP3
PROTEG_MP3	21	Protección MP3
LAYER_MP3	22	Capa MP3
FREC_MP3	23	Frecuencia MP3
RELLENO_MP3	24	Bits de relleno MP3
MODO_MP3	25	Canal MP3
COPY_MP3	26	Copyright MP3
ORIGINAL_MP3	27	MP3 Original
ENFASIS_MP3	28	Enfasis MP3
FREC_FLAC	29	Frecuencia de muestreo FLAC
CANAL_FLAC	30	Canal de FLAC
BITSMUESTA_FLAC	31	Bits por muestra FLAC
VERSION_MPEG	32	Versión MPEG
LAYER_MPEG	33	Layer MPEG
PROTEG_MPEG	34	Protección MPEG
BITRATE_MPEG	35	Bitrate MPEG
FREC_MPEG	36	Frecuencia MPEG
RELLENO_MPEG	37	Relleno MPEG
MODO_MPEG	38	Modo MPEG
COPY_MPEG	39	Copyright MPEG
ORIGINAL_MPEG	40	Original MPEG
ENFASIS_MPEG	41	Énfasis MPEG
ANCHURA_ELEM	43	Anchura de Elementary Stream
ALTURA_ELEM	44	Altura de Elementary Stream
ASPEC_ELEM	45	Aspecto Elementary Stream
FRAMERATE_ELEM	46	Framerate Elementary Stream
BITRATE_ELEM	47	Bitrate de Elementary Stream
V1_mp3	1	Versiones MP3
V2_mp3	0	
V1_mpeg	1	Versiones MPEG
V2_mpeg	0	
BI_RGB	0	Compresiones BMP
BI_RLE8	1	
BI_RLE4	2	
BI_BITFIELDS	3	
BI_JPEG	4	
BI_PNG	5	

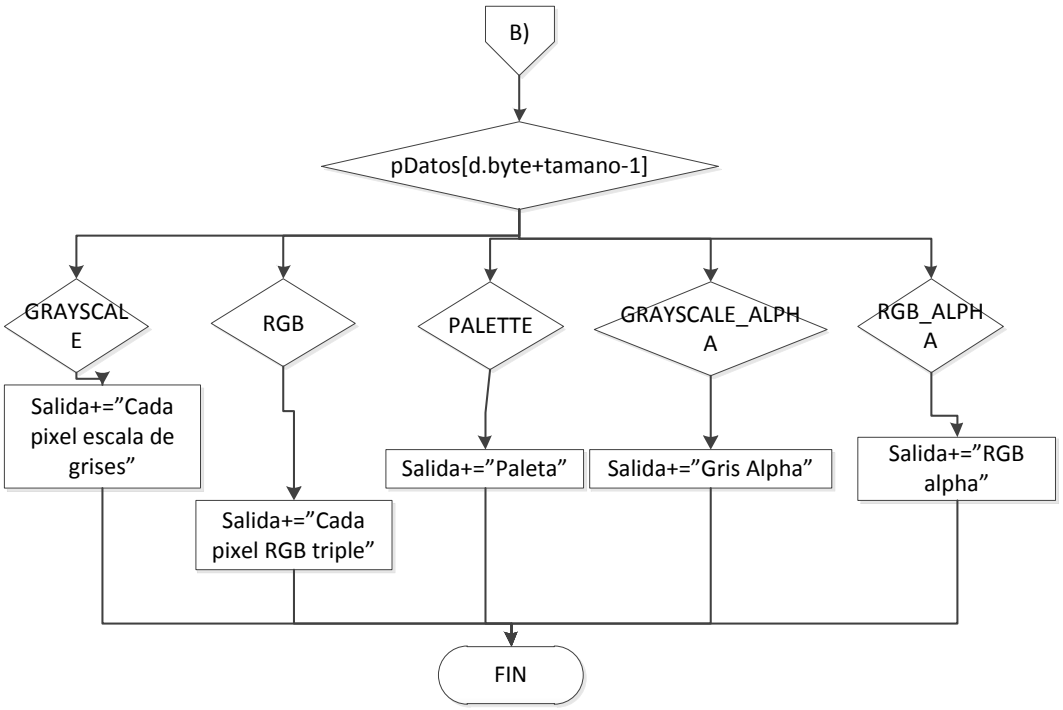
BI_ALPHABITFIELDS	6	
GRAYSCALE	0	Color PNG
RGB	2	
PALETTE	3	
GRAYSCALE_ALPHA	4	
RGB_ALPHA	6	
DEFECTO	0	Compresion PNG
NINGUNO	0	Filtrado PNG
SUB	1	
UP	2	
AVERAGE	3	
PAETH	4	
NO_ENTRELAZADO	0	Entrelazado PNG
ADAM7	1	
V25	0	Versión PCX
V28PALETA	2	
V28PALETA_POR_DEFECTO	3	
PAINTBRUSH	4	
V30	5	
RLE	1	Codificación PCX
MONOCRONO	1	Bits/pixel PCX
COLORES16	4	
COLORES256	8	
TRUECOLOR	24	
COLOR	1	Color PCX
ESCALA_GRISES	2	
MONOCRONO_PSD	0	Color PSD
ESCALA_GRISES_PSD	1	
PALETA_COLOR_PSD	2	
RGB_PSD	3	
CMYK_PSD	4	
MULTICANAL_PSD	7	
DUOTONE_PSD	8	
LAB_PSD	9	
SIN_UNIDADES	0	Unidades JPEG
PUNTOS_PULGADAS	1	
PUNTOS_CM	2	
UN_TRACK	0	Formato MIDI
DOS_TRACKS	1	
MULTIPLES_TRACKS	2	

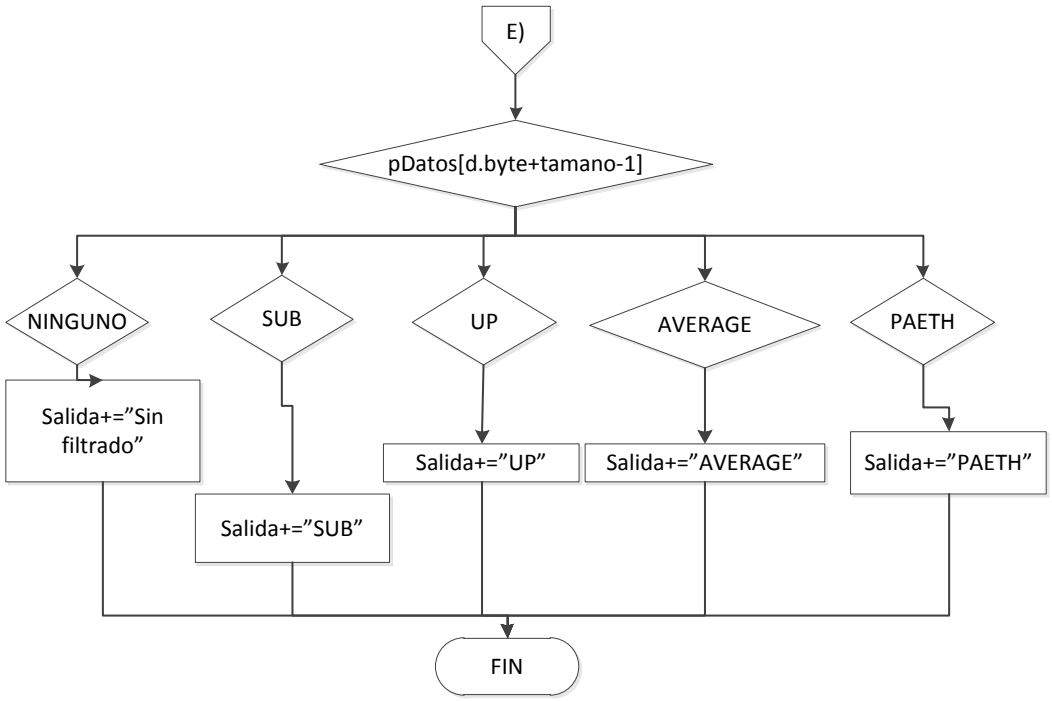
VIDEO	1	Bitmask FLV
AUDIO	4	
AUDIO_VIDEO	5	
TIPO_VIDEO	9	Tipo FLV
TIPO_AUDIO	8	
TIPO_META	12	
IPPM	2	Tipo PPM/PGM/PBM
IPPM2	3	
IPGM	8	
IPGM2	9	
IPBM	10	
IPBM2	31	
IRIFF	35	Tipo de RIFF
IWAV	34	
IAVI	19	

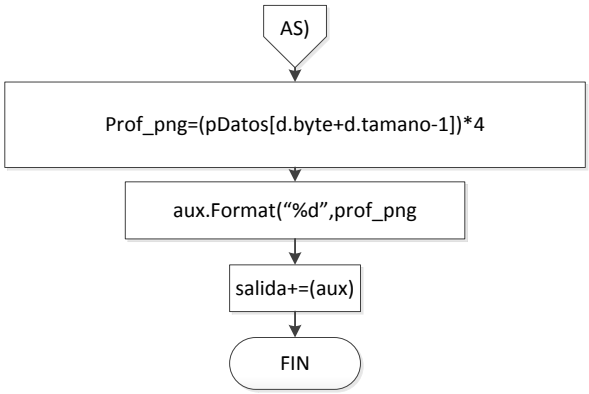
Tabla 56 Constantes

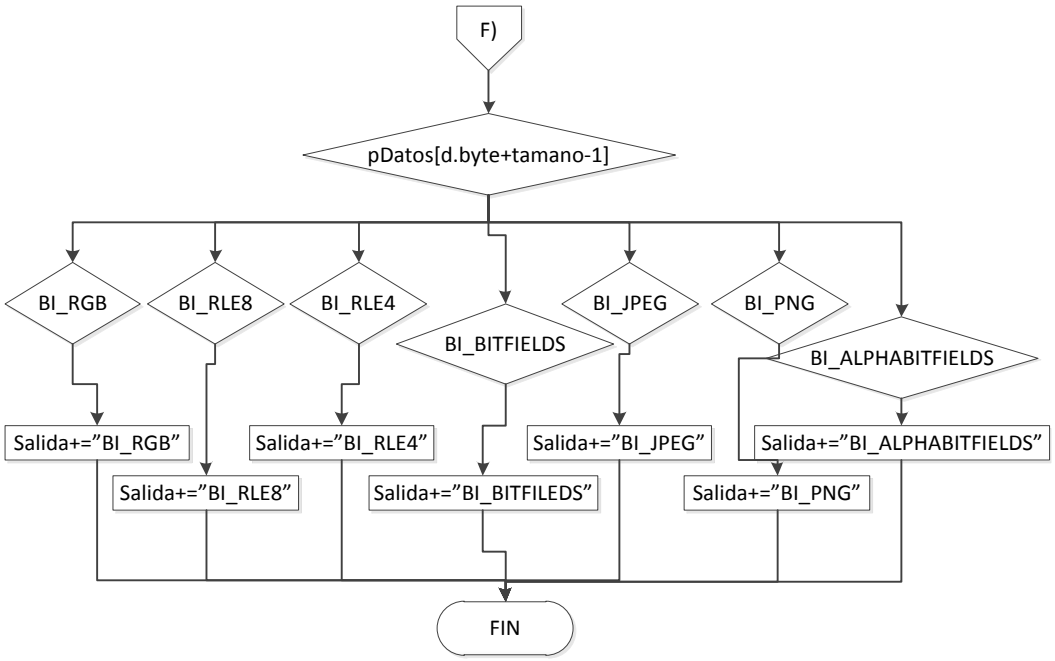
3.2.5.1. Desglose de la función **GenerarCadenaDato**

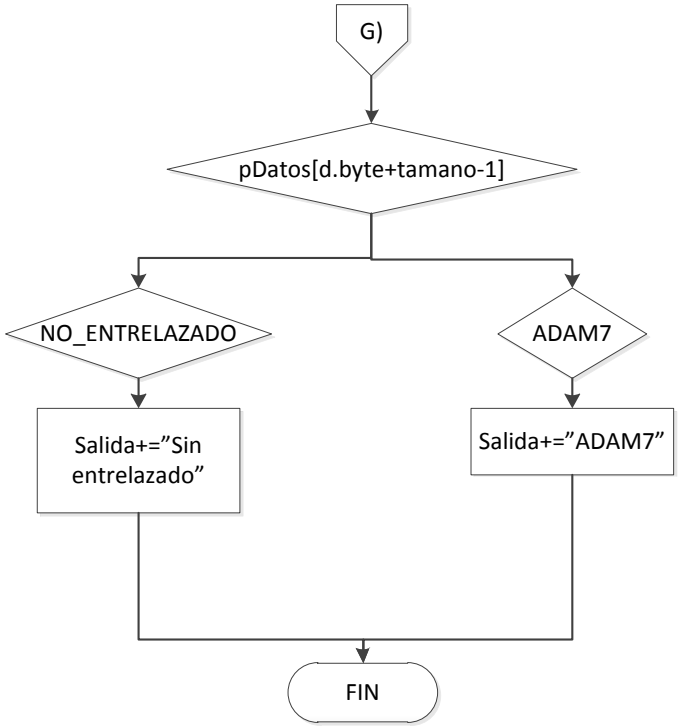
A continuación se presenta el desglose de la función GenerarCadenaDato para cada uno de los casos posibles que se vayan a analizar.

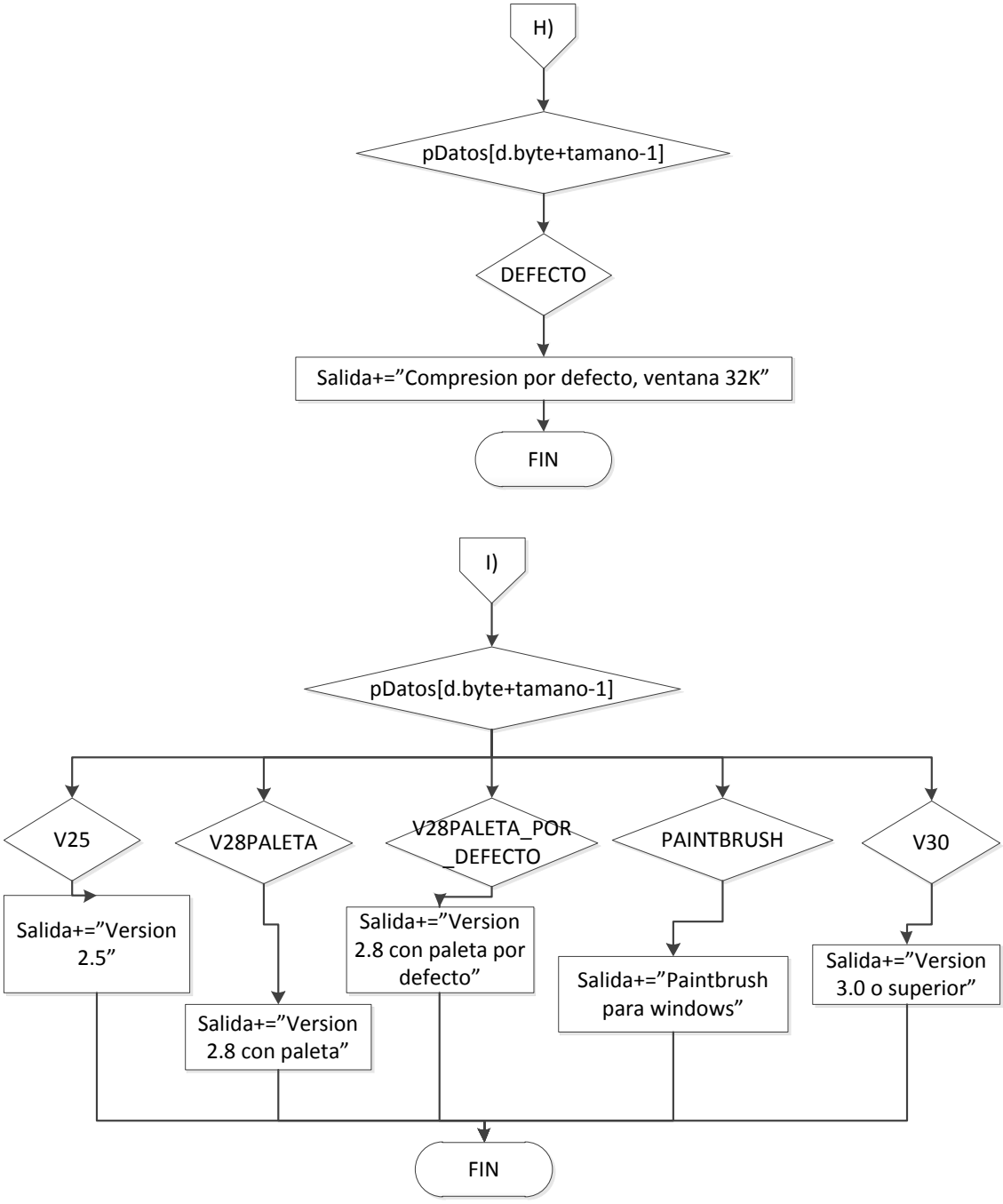


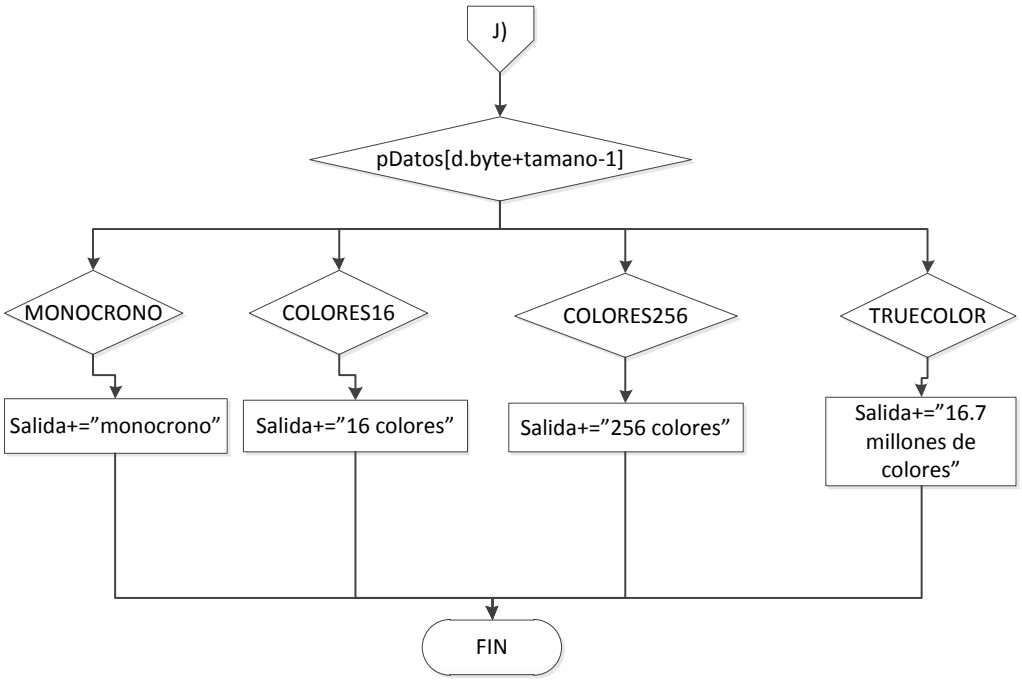

E)
pDatos[d.byte+tamano-1]
NINGUNO
SUB
UP
AVERAGE
PAETH
Salida+=”Sin filtrado”
Salida+=”SUB”
Salida+=”UP”
Salida+=”AVERAGE”
Salida+=”PAETH”
FIN

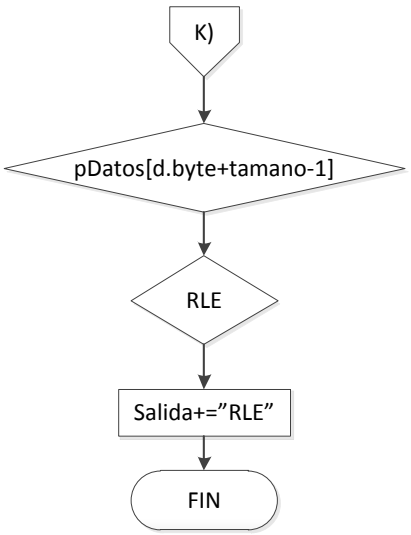

AS)
Prof_png=(pDatos[d.byte+d.tamano-1])*4
aux.Format(“%d”,prof_png
salida+=(aux)
FIN

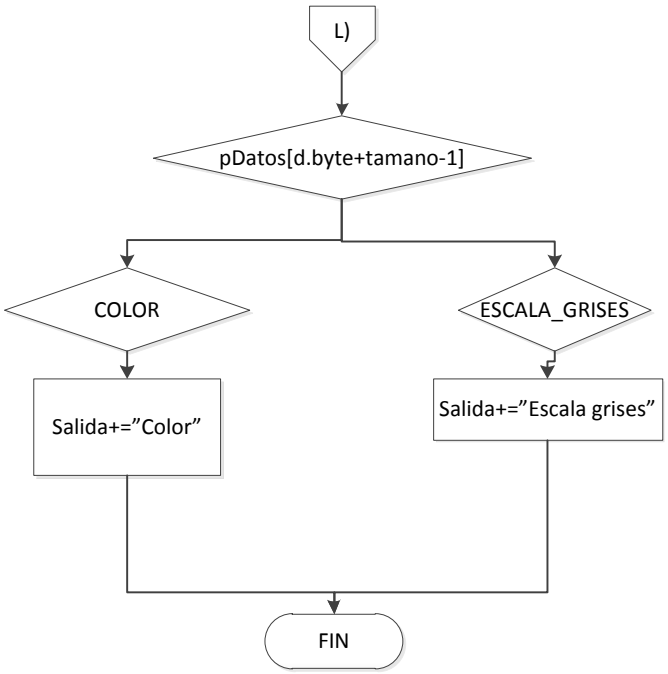

F)
pDatos[d.byte+tamano-1]
BI_RGB
BI_RLE8
BI_RLE4
BI_BITFIELDS
BI_JPEG
BI_PNG
BI_ALPHABITFIELDS
Salida+=”BI_RGB”
Salida+=”BI_RLE8”
Salida+=”BI_RLE4”
Salida+=”BI_BITFILEDS”
Salida+=”BI_JPEG”
Salida+=”BI_PNG”
Salida+=”BI_ALPHABITFIELDS”
FIN

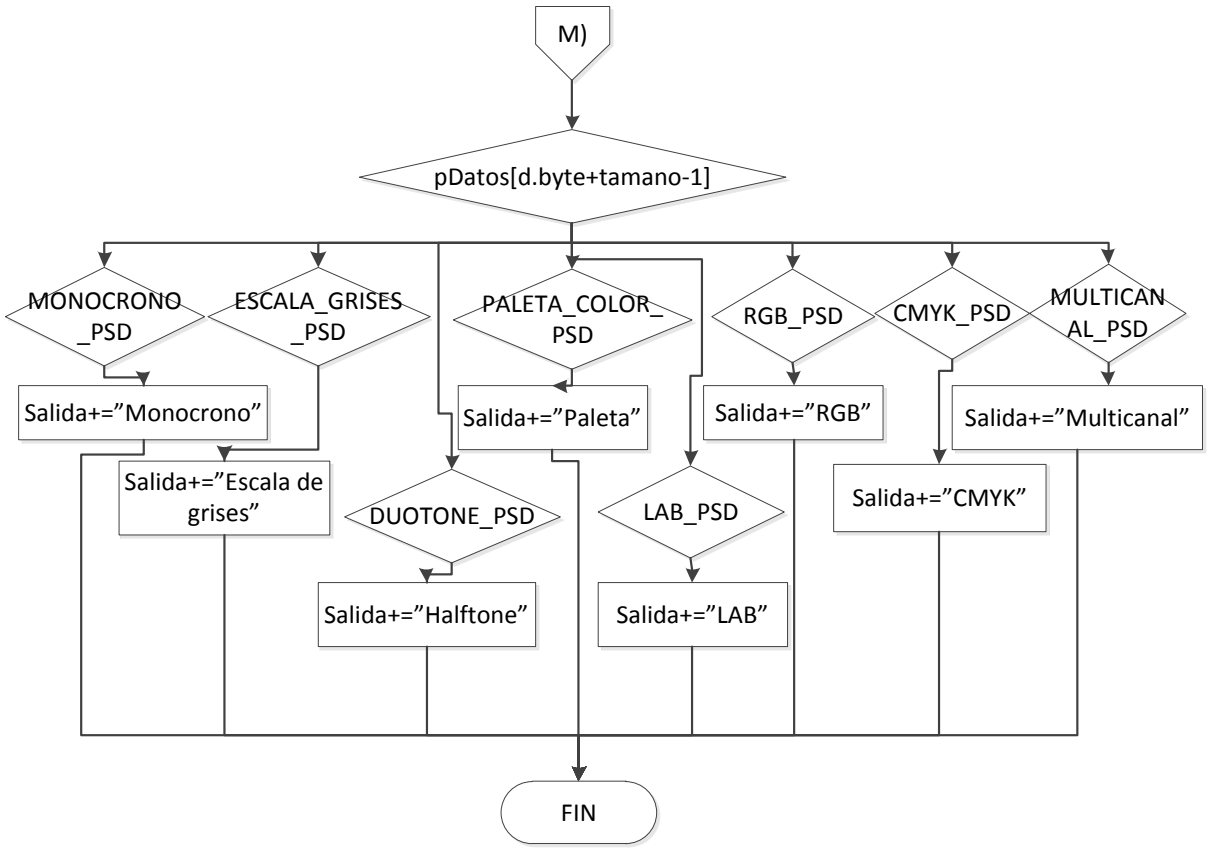

G)
pDatos[d.byte+tamano-1]
NO_ENTRELAZADO
ADAM7
Salida+=”Sin entrelazado”
Salida+=”ADAM7”
FIN

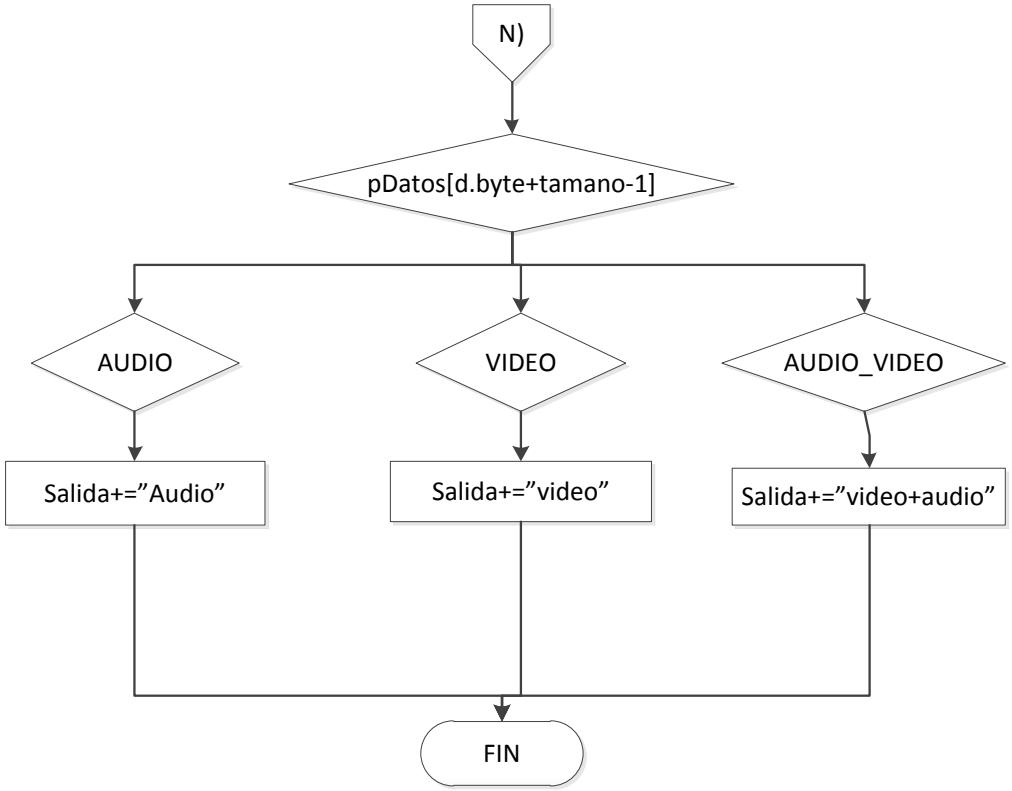

H)
pDatos[d.byte+tamano-1]
DEFECTO
Salida+=”Compresion por defecto, ventana 32K”
FIN
I)
pDatos[d.byte+tamano-1]
V25
V28PALETA
V28PALETA_POR_DEFECTO
PAINTBRUSH
V30
Salida+=”Version 2.5”
Salida+=”Version 2.8 con paleta”
Salida+=”Version 2.8 con paleta por defecto”
Salida+=”Paintbrush para windows”
Salida+=”Version 3.0 o superior”
FIN

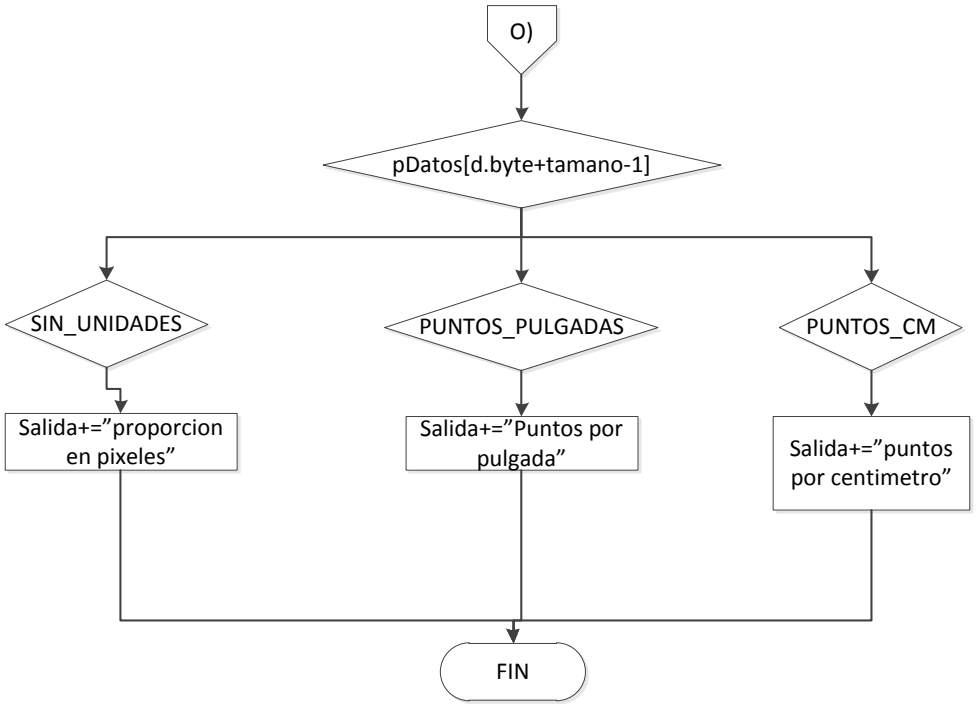

J)
pDatos[d.byte+tamano-1]
MONOCRONO
COLORES16
COLORES256
TRUECOLOR
Salida+=”monocrono”
Salida+=”16 colores”
Salida+=”256 colores”
Salida+=”16.7 millones de colores”
FIN

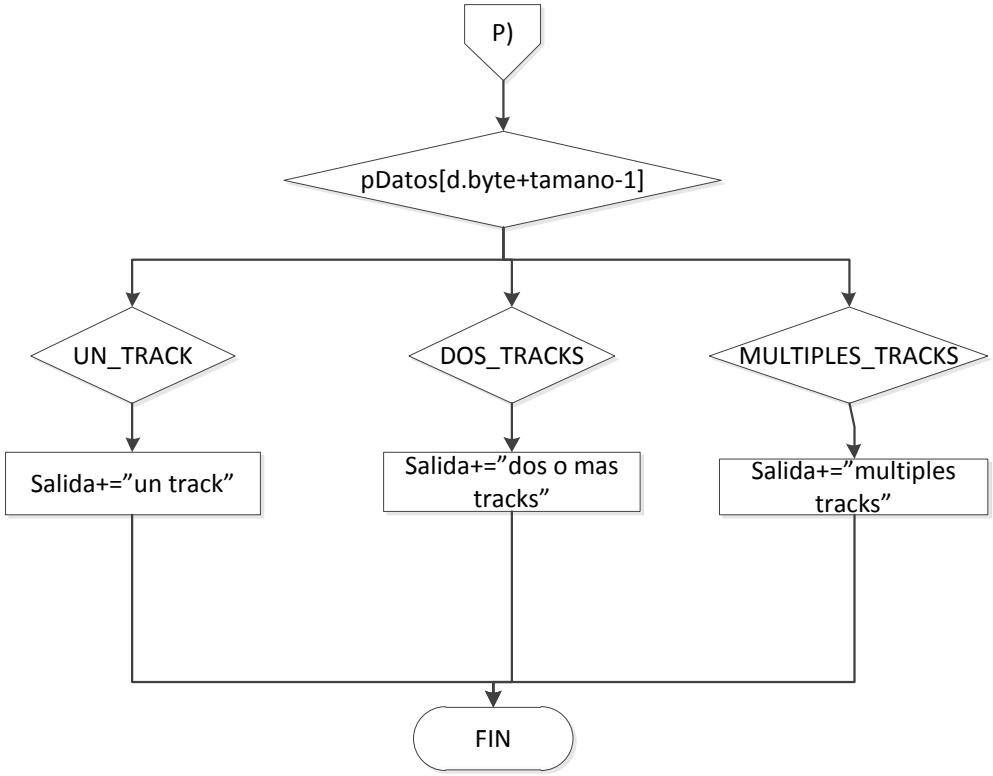

K)
pDatos[d.byte+tamano-1]
RLE
Salida+=”RLE”
FIN

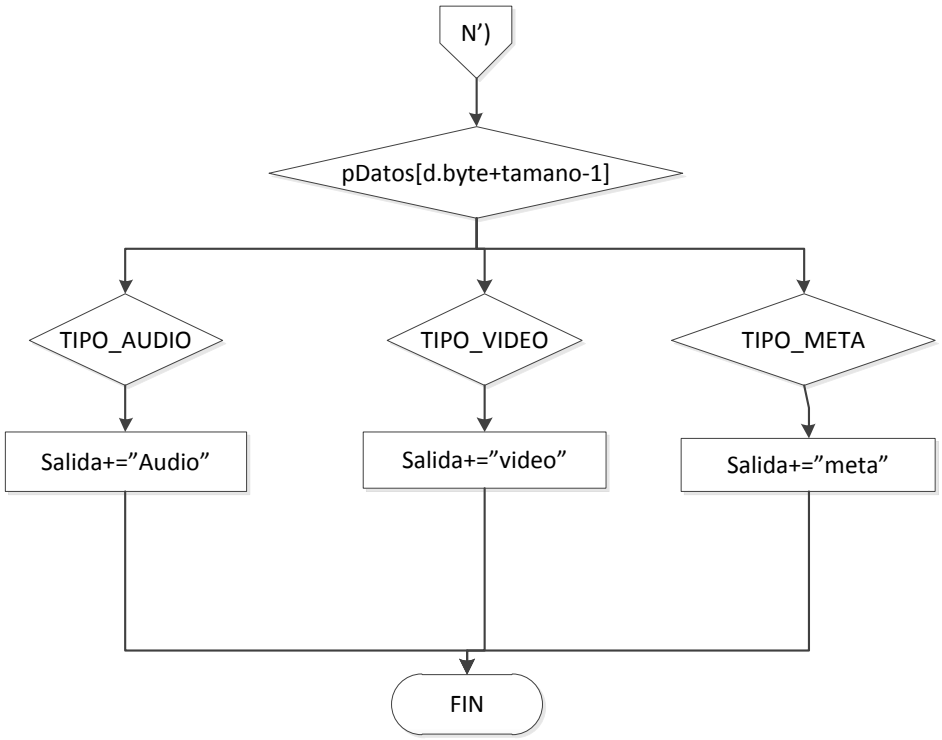

L)
pDatos[d.byte+tamano-1]
COLOR
ESCALA_GRISES
Salida+=”Color”
Salida+=”Escala grises”
FIN

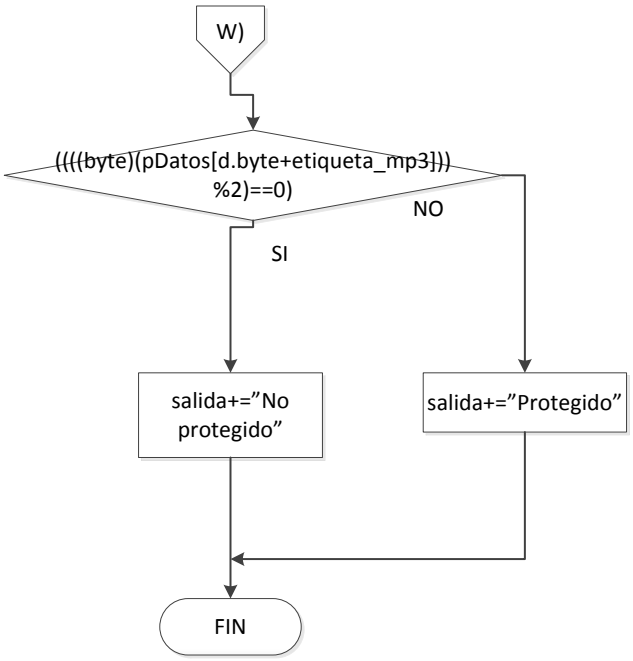

M)
pDatos[d.byte+tamano-1]
MONOCRONO_PSD
ESCALA_GRISES_PSD
PALETA_COLOR_PSD
RGB_PSD
CMYK_PSD
MULTICANAL_PSD
Salida+=”Monocrono”
Salida+=”Escala de grises”
Salida+=”Paleta”
DUOTONE_PSD
Salida+=”Halftone”
LAB_PSD
Salida+=”LAB”
Salida+=”RGB”
Salida+=”CMYK”
Salida+=”Multicanal”
FIN

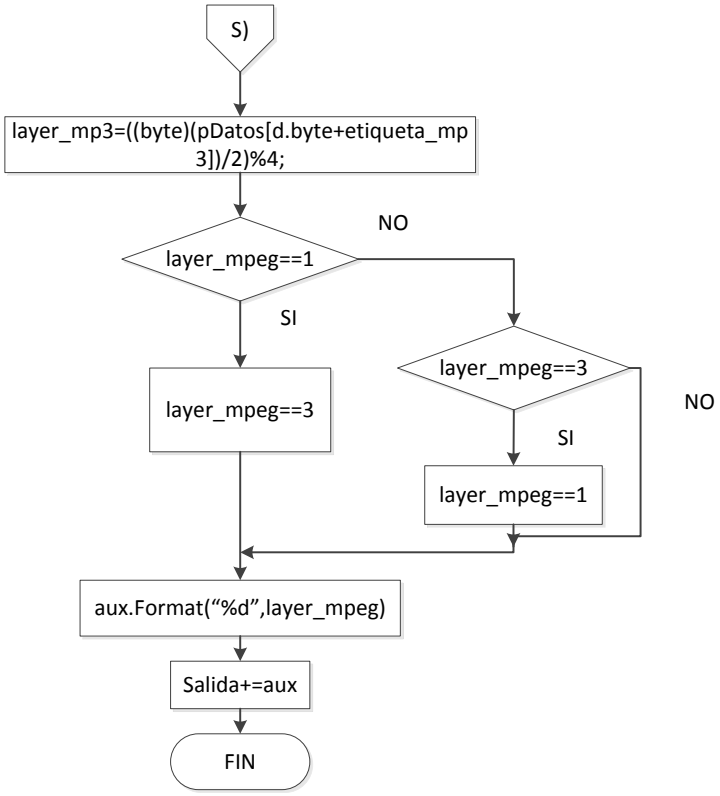

N)
pDatos[d.byte+tamano-1]
AUDIO
VIDEO
AUDIO_VIDEO
Salida+=”Audio”
Salida+=”video”
Salida+=”video+audio”
FIN

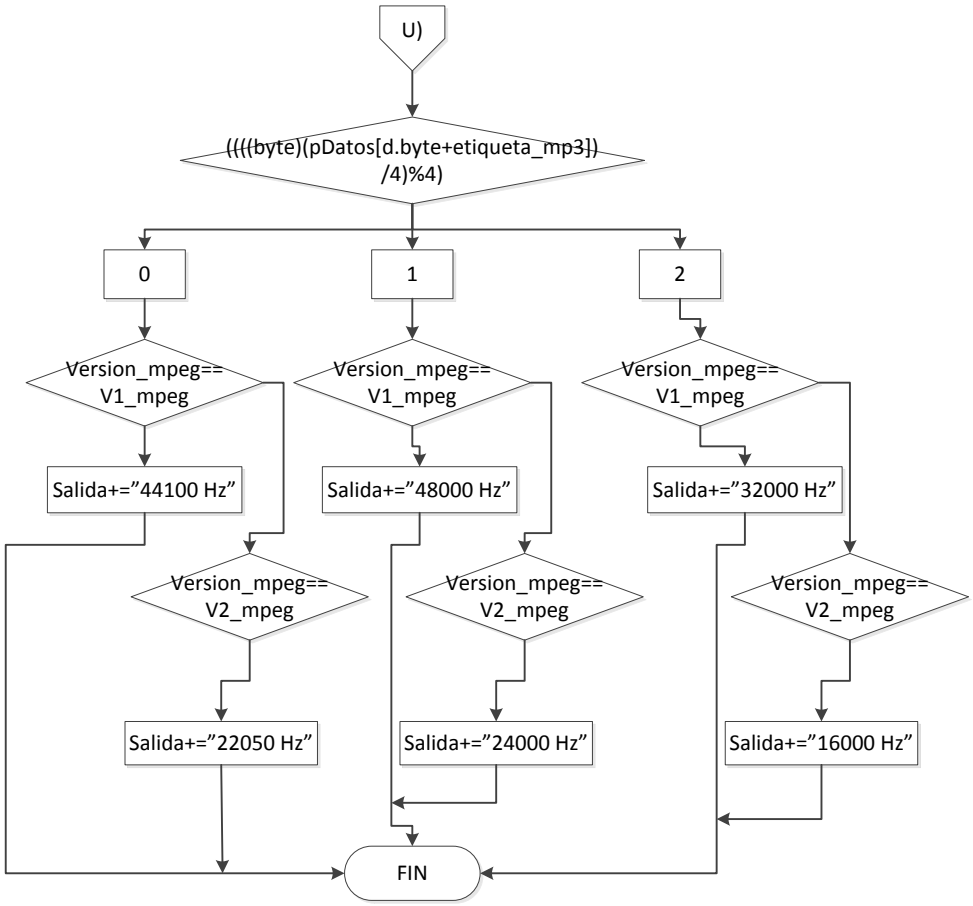

O)
pDatos[d.byte+tamano-1]
SIN_UNIDADES
PUNTOS_PULGADAS
PUNTOS_CM
Salida+=”proporcion en pixeles”
Salida+=”Puntos por pulgada”
Salida+=”puntos por centimetro”
FIN

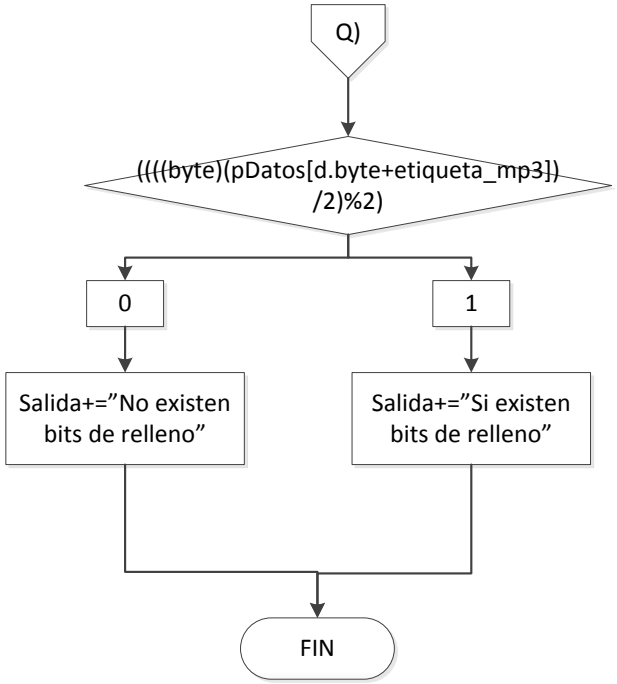

P)
pDatos[d.byte+tamano-1]
UN_TRACK
DOS_TRACKS
MULTIPLES_TRACKS
Salida+=”un track”
Salida+=”dos o mas tracks”
Salida+=”multiples tracks”
FIN

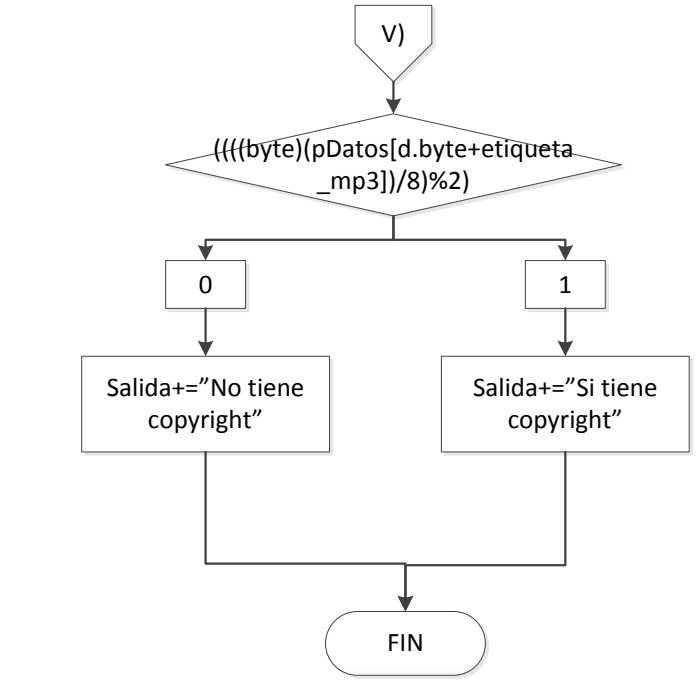

N’)
pDatos[d.byte+tamano-1]
TIPO_AUDIO
TIPO_VIDEO
TIPO_META
Salida+=”Audio”
Salida+=”video”
Salida+=”meta”
FIN

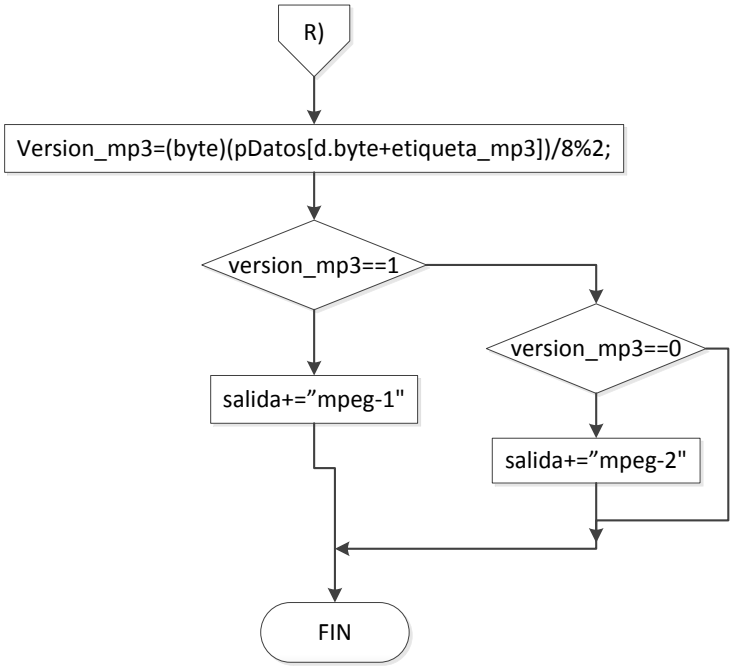

W)
((((byte)(pDatos[d.byte+etiqueta_mp3]))
%2)==0)
NO
SI
salida+=”No
protegido”
salida+=”Protegido”
FIN

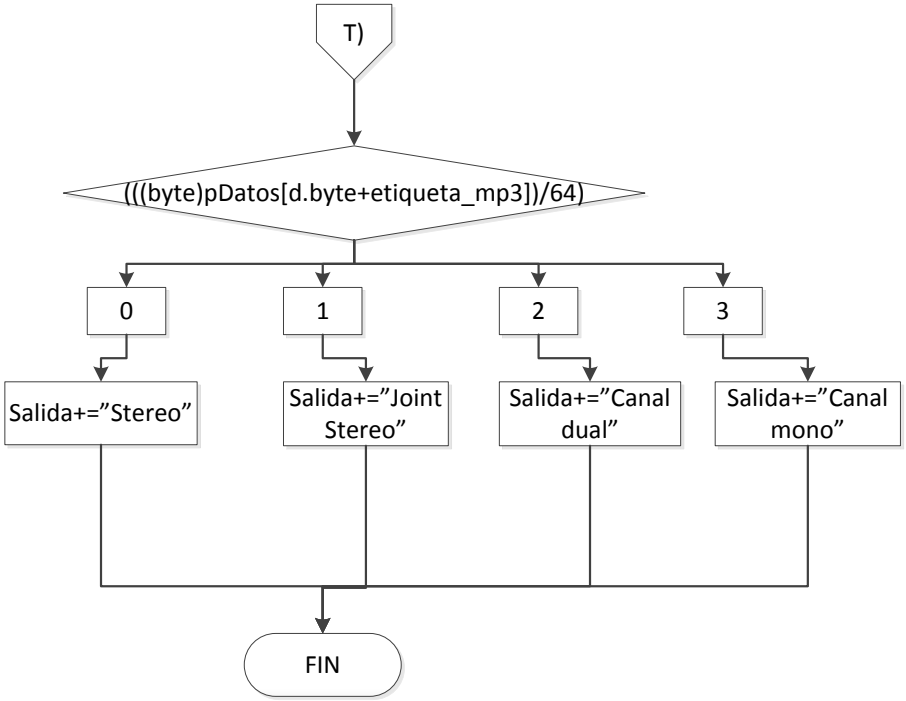

S)
layer_mp3=((byte)(pDatos[d.byte+etiqueta_mp
3])/2)%4;
NO
layer_mpeg==1
SI
layer_mpeg==3
layer_mpeg==3
NO
SI
layer_mpeg==1
aux.Format(“%d”,layer_mpeg)
Salida+=aux
FIN

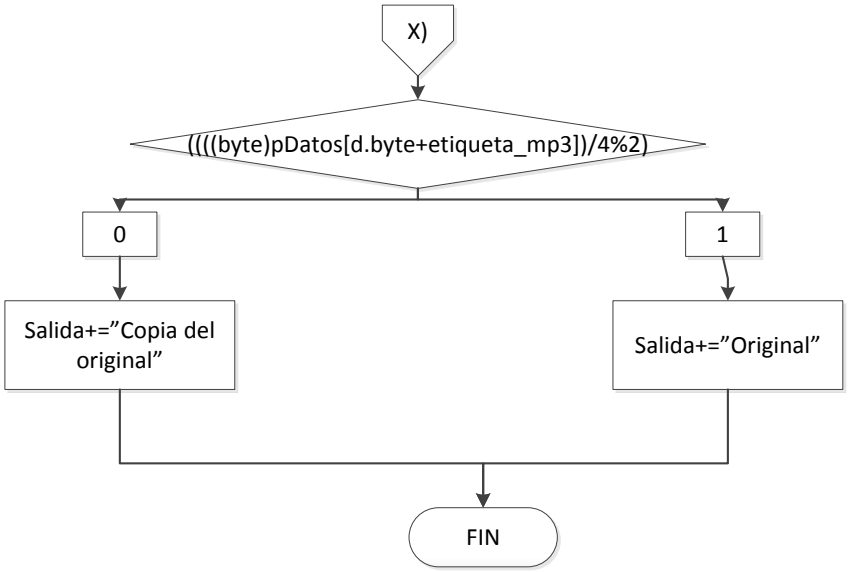

U)
((((byte)(pDatos[d.byte+etiqueta_mp3])
/4)%4)
0
1
2
Version_mpeg==
V1_mpeg
Version_mpeg==
V1_mpeg
Version_mpeg==
V1_mpeg
Salida+=”44100 Hz”
Salida+=”48000 Hz”
Salida+=”32000 Hz”
Version_mpeg==
V2_mpeg
Version_mpeg==
V2_mpeg
Version_mpeg==
V2_mpeg
Salida+=”22050 Hz”
Salida+=”24000 Hz”
Salida+=”16000 Hz”
FIN

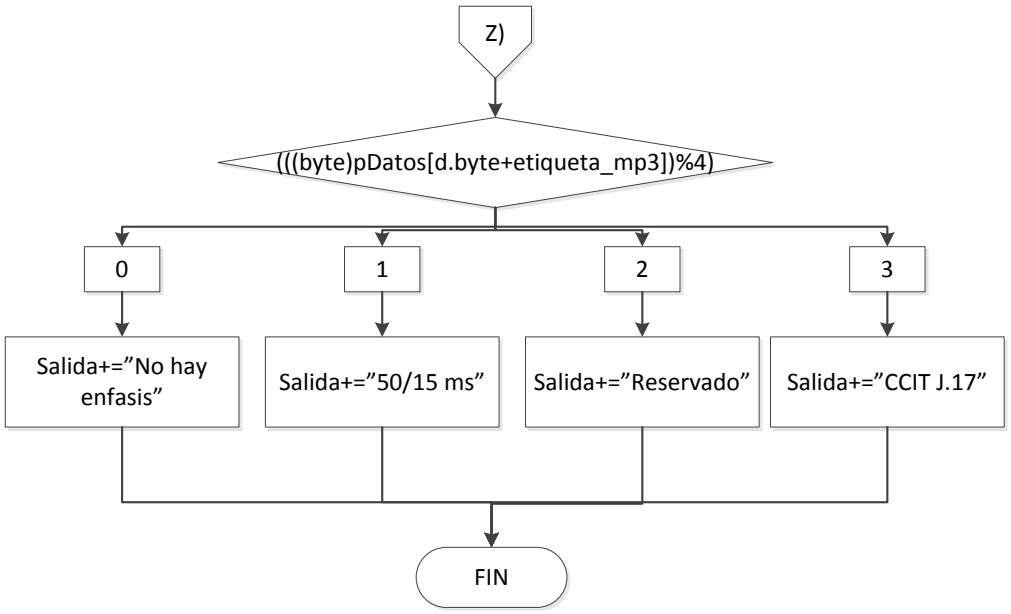

Q)
((((byte)(pDatos[d.byte+etiqueta_mp3])
/2)%2)
0
1
Salida+=”No existen
bits de relleno”
Salida+=”Si existen
bits de relleno”
FIN

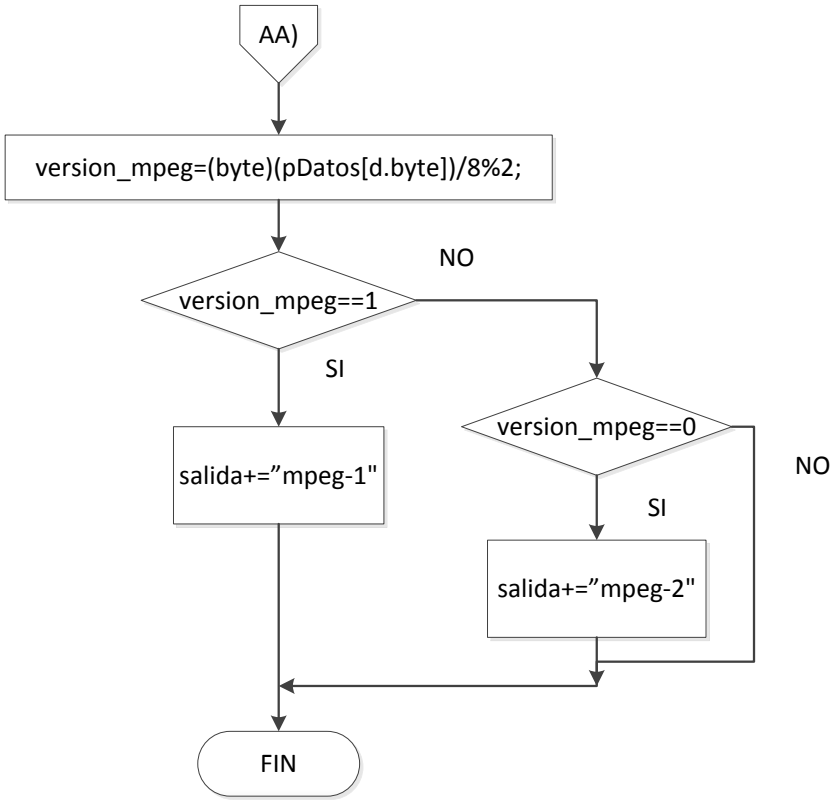

V)
((((byte)(pDatos[d.byte+etiqueta_mp3])/8)%2)
0
1
Salida+=”No tiene copyright”
Salida+=”Si tiene copyright”
FIN

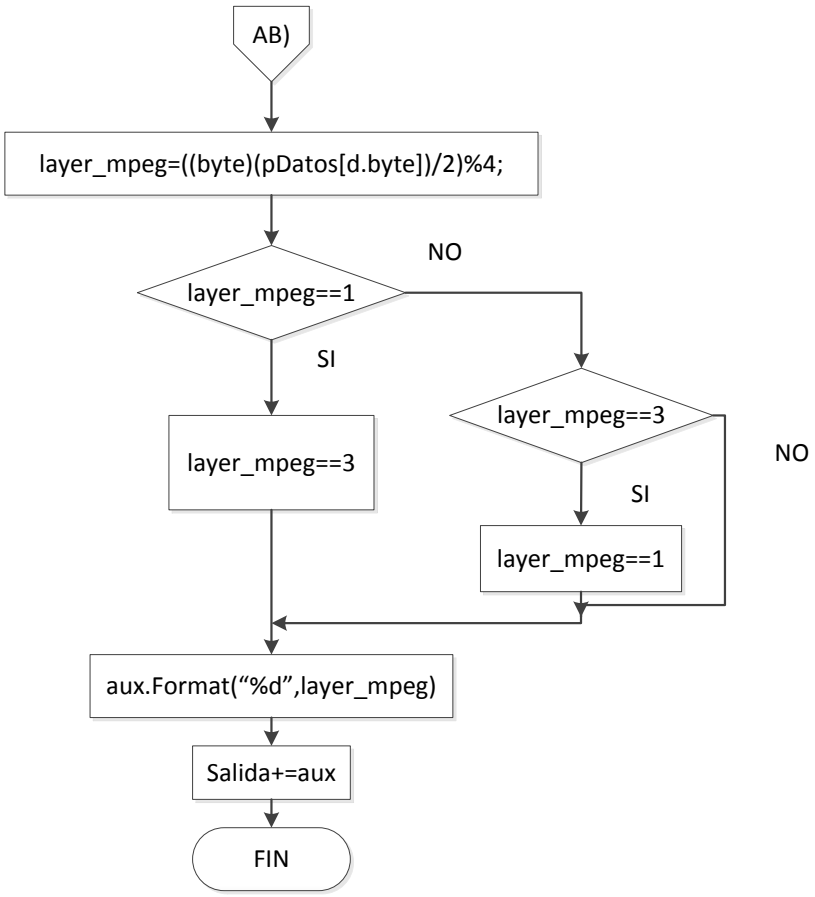

R)
Version_mp3=(byte)(pDatos[d.byte+etiqueta_mp3])/8%2;
version_mp3==1
version_mp3==0
salida+=”mpeg-1"
salida+=”mpeg-2"
FIN

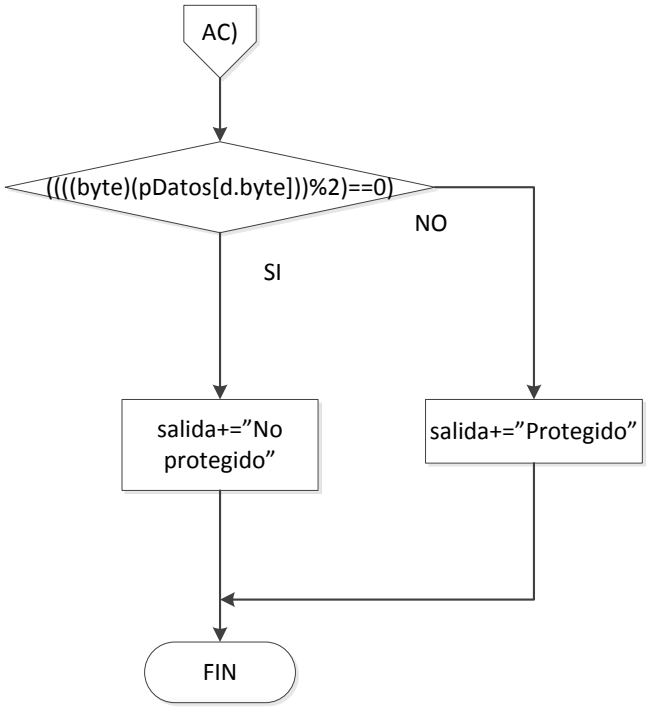

T)
(((byte)pDatos[d.byte+etiqueta_mp3])/64)
0
1
2
3
Salida+=”Stereo”
Salida+=”Joint Stereo”
Salida+=”Canal dual”
Salida+=”Canal mono”
FIN

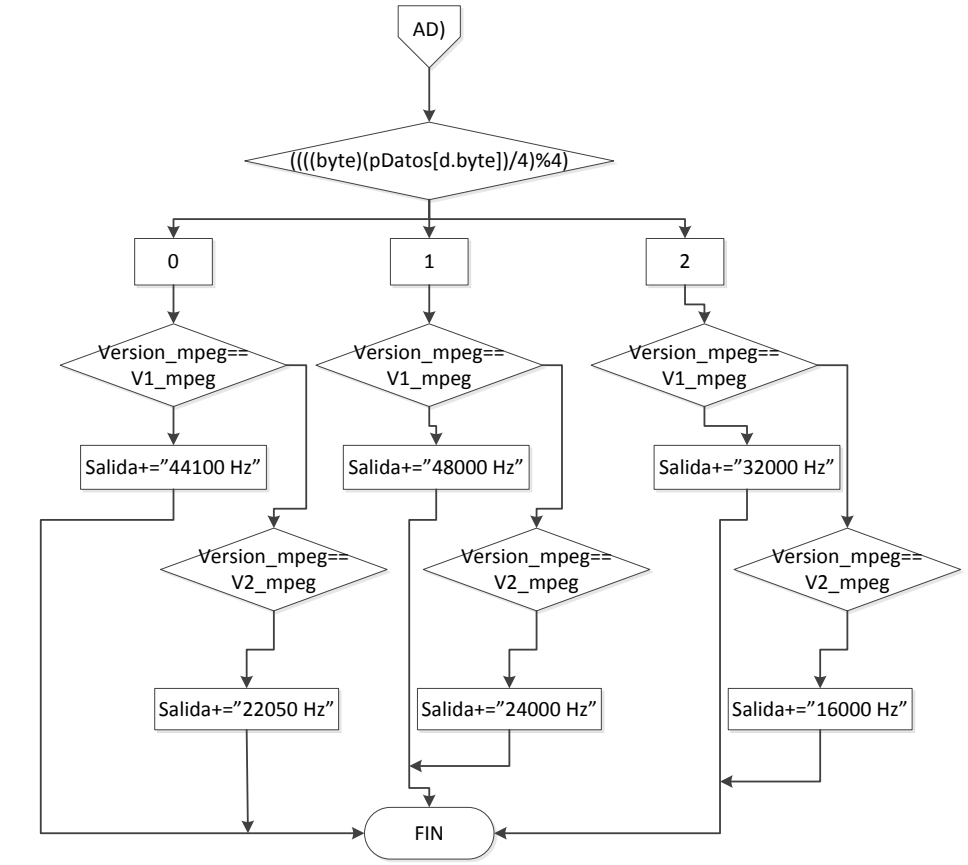

X)
((((byte)pDatos[d.byte+etiqueta_mp3])/4%2)
0
1
Salida+=”Copia del original”
Salida+=”Original”
FIN

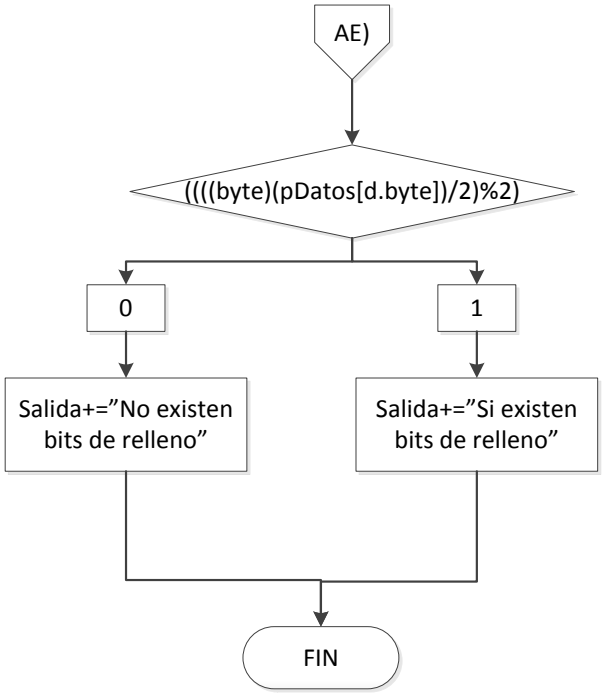

Z)
(((byte)pDatos[d.byte+etiqueta_mp3])%4)
0
1
2
3
Salida+=”No hay enfasis”
Salida+=”50/15 ms”
Salida+=”Reservado”
Salida+=”CCIT J.17”
FIN

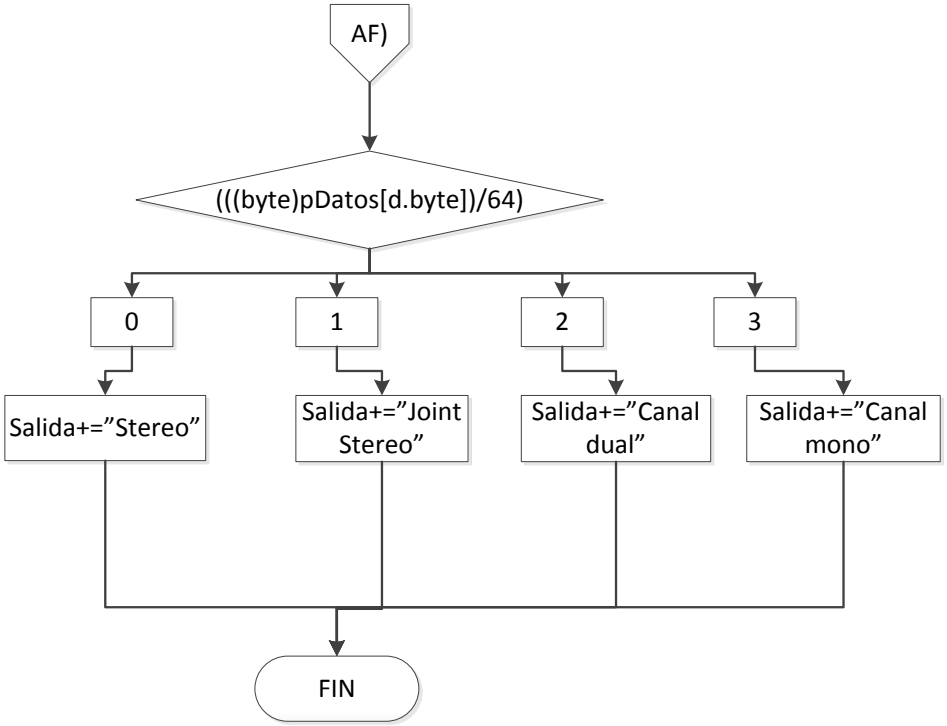

AA)
version_mpeg=(byte)(pDatos[d.byte])/8%2;
version_mpeg==1
NO
SI
salida+=”mpeg-1"
version_mpeg==0
NO
SI
salida+=”mpeg-2"
FIN

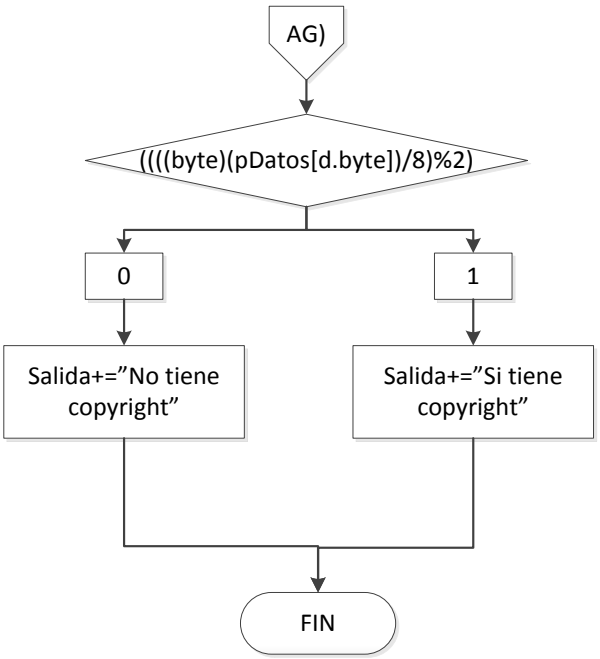

AB)
layer_mpeg=((byte)(pDatos[d.byte])/2)%4;
layer_mpeg==1
NO
SI
layer_mpeg==3
layer_mpeg==3
NO
SI
layer_mpeg==1
aux.Format("%d",layer_mpeg)
Salida+=aux
FIN

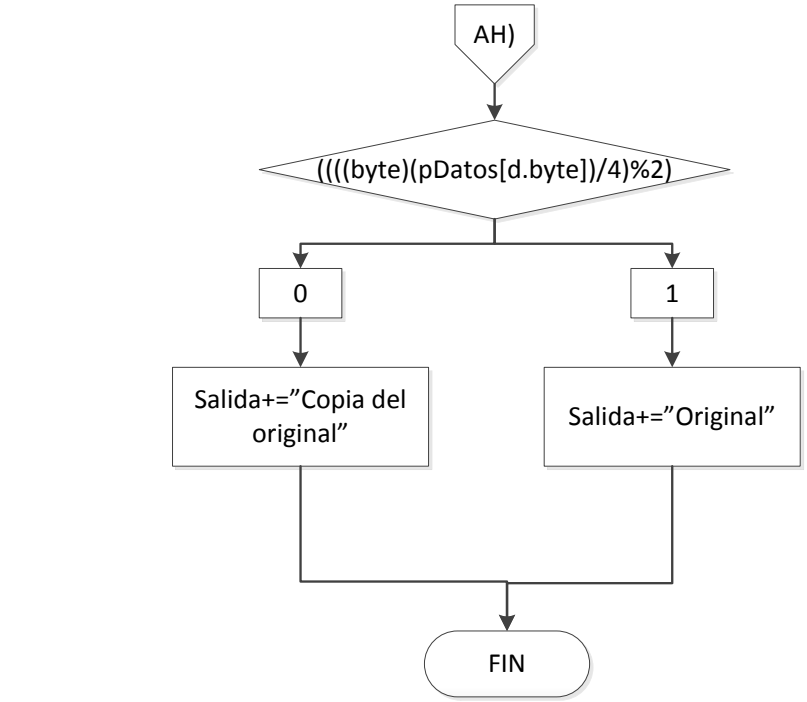

AC)
((((byte)(pDatos[d.byte]))%2)==0)
NO
SI
salida+="No protegido"
salida+="Protegido"
FIN

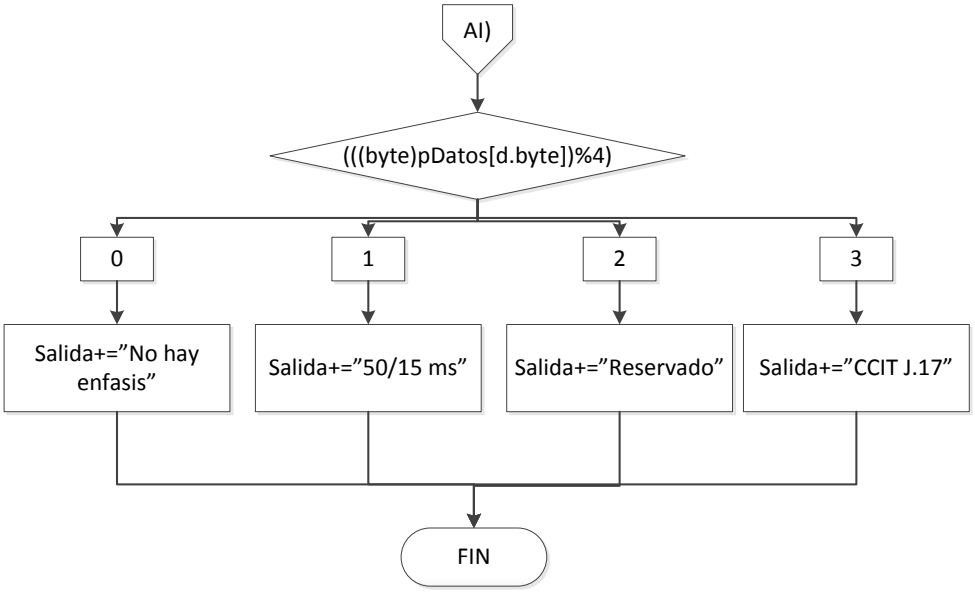

AD)
((((byte)(pDatos[d.byte])/4)%4)
0
1
2
Version_mpeg== V1_mpeg
Version_mpeg== V1_mpeg
Version_mpeg== V1_mpeg
Salida+=”44100 Hz”
Salida+=”48000 Hz”
Salida+=”32000 Hz”
Version_mpeg== V2_mpeg
Version_mpeg== V2_mpeg
Version_mpeg== V2_mpeg
Salida+=”22050 Hz”
Salida+=”24000 Hz”
Salida+=”16000 Hz”
FIN

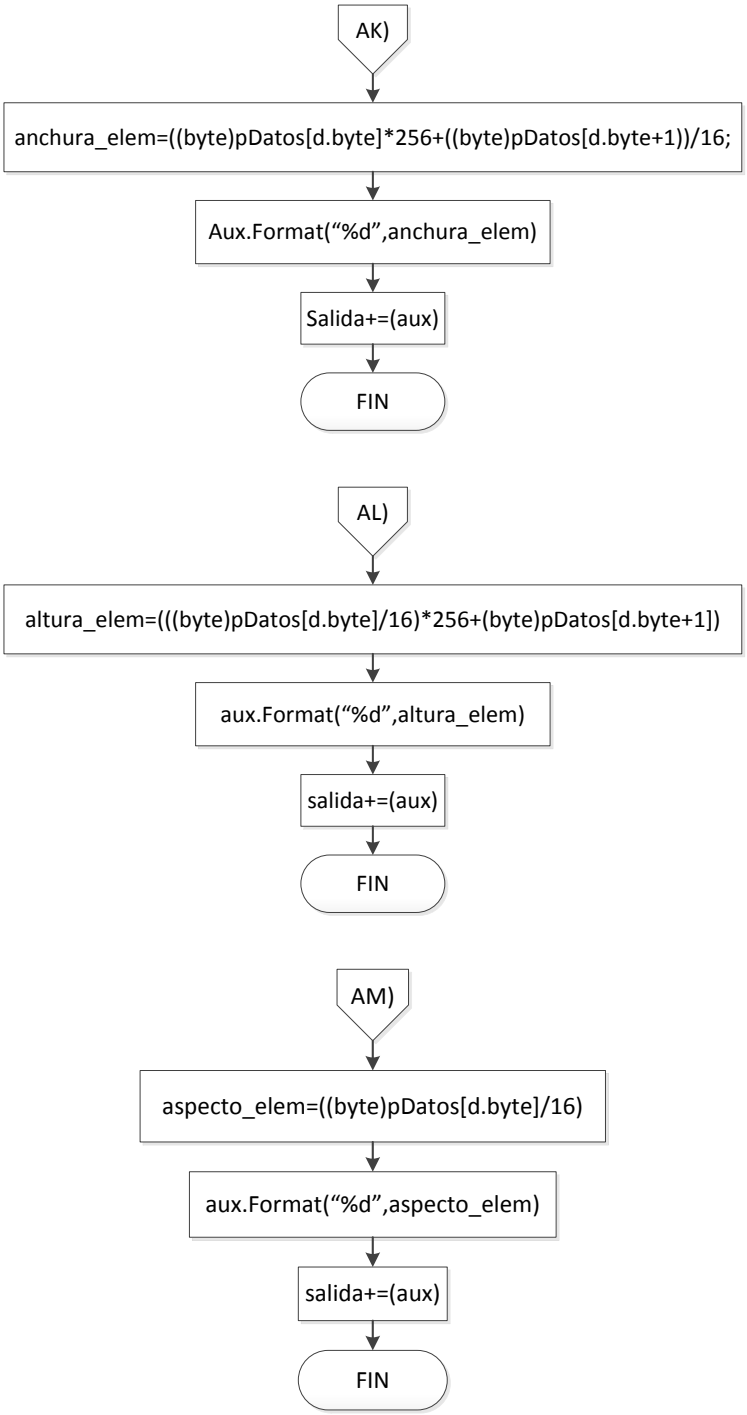

AE)
((((byte)(pDatos[d.byte])/2)%2)
0
1
Salida+=”No existen bits de relleno”
Salida+=”Si existen bits de relleno”
FIN

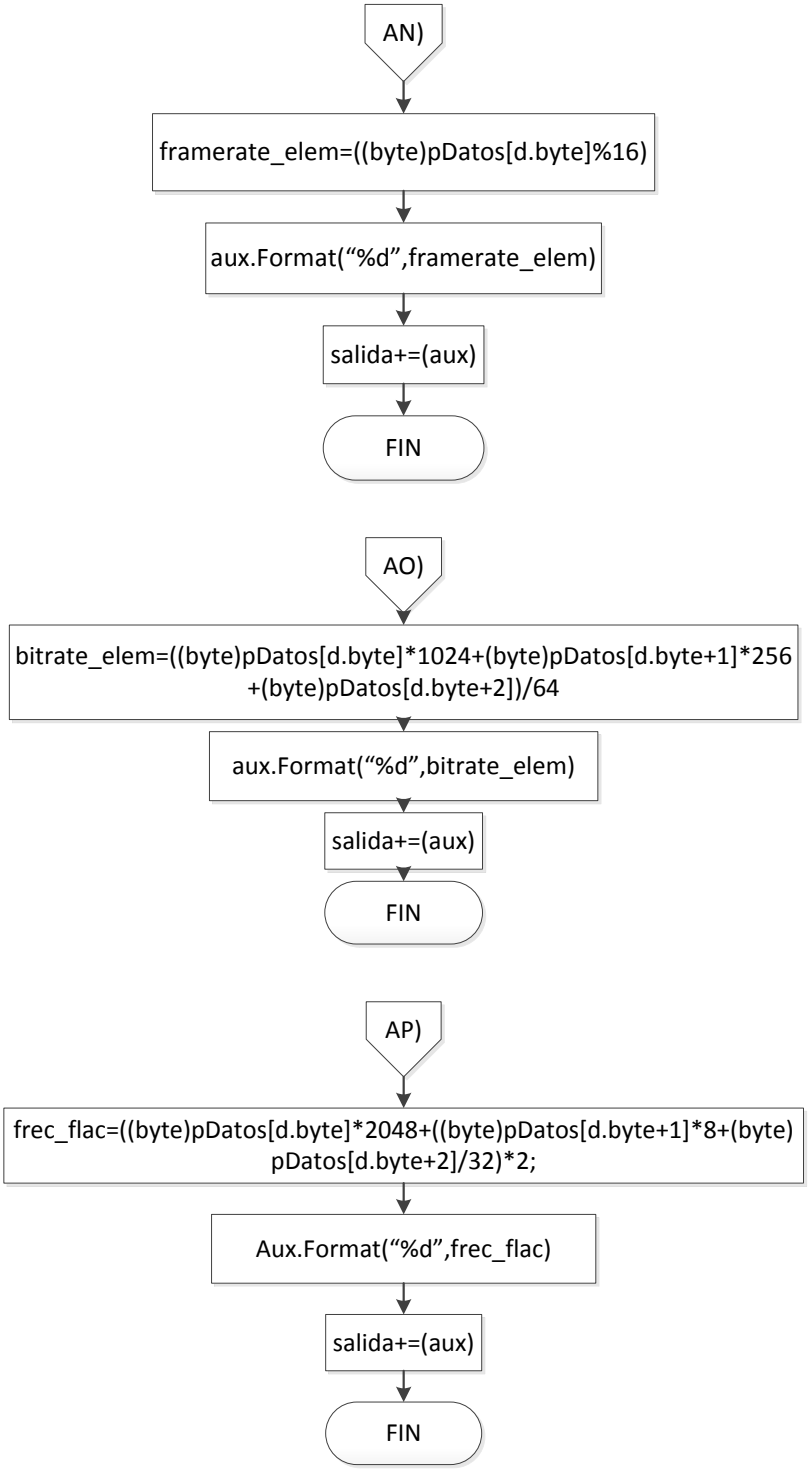

AF)
(((byte)pDatos[d.byte])/64)
0
1
2
3
Salida+=”Stereo”
Salida+=”Joint Stereo”
Salida+=”Canal dual”
Salida+=”Canal mono”
FIN

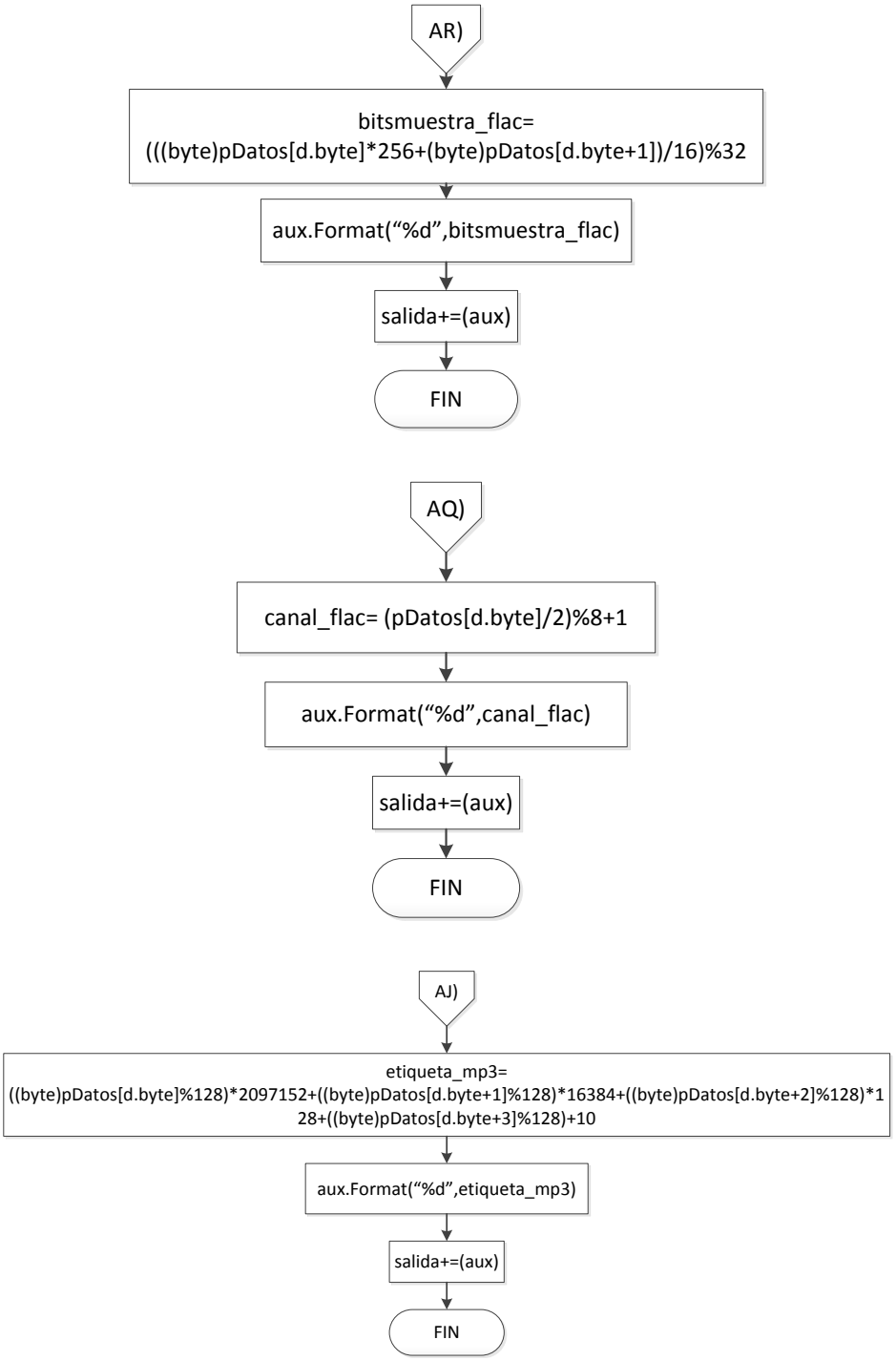

AG)
((((byte)(pDatos[d.byte])/8)%2)
0
1
Salida+=”No tiene copyright”
Salida+=”Si tiene copyright”
FIN


AH)
((((byte)(pDatos[d.byte])/4)%2)
0
1
Salida+=”Copia del original”
Salida+=”Original”
FIN


AI)
(((byte)pDatos[d.byte])%4)
0
1
2
3
Salida+=”No hay enfasis”
Salida+=”50/15 ms”
Salida+=”Reservado”
Salida+=”CCIT J.17”
FIN


AK)
anchura_elem=((byte)pDatos[d.byte]*256+((byte)pDatos[d.byte+1))/16;
Aux.Format("%d",anchura_elem)
Salida+=(aux)
FIN
AL)
altura_elem=(((byte)pDatos[d.byte]/16)*256+(byte)pDatos[d.byte+1])
aux.Format("%d",altura_elem)
salida+=(aux)
FIN
AM)
aspecto_elem=((byte)pDatos[d.byte]/16)
aux.Format("%d",aspecto_elem)
salida+=(aux)
FIN


AN)
framerate_elem=((byte)pDatos[d.byte]%16)
aux.Format("%d",framerate_elem)
salida+=(aux)
FIN
AO)
bitrate_elem=((byte)pDatos[d.byte]*1024+(byte)pDatos[d.byte+1]*256
+(byte)pDatos[d.byte+2])/64
aux.Format("%d",bitrate_elem)
salida+=(aux)
FIN
AP)
frec_flac=((byte)pDatos[d.byte]*2048+((byte)pDatos[d.byte+1]*8+(byte)
pDatos[d.byte+2]/32)*2;
Aux.Format("%d",frec_flac)
salida+=(aux)
FIN



Como se comentó en el apartado de los formatos, en los formatos PPM, PGM y PBM pueden contener uno o varios comentarios y de tamaños no específicos entre los bytes de identificación y los de información. Si existen dichos comentarios, el usuario puede leer esta información perfectamente dado que viene expresada en ASCII pero el resto de información quedará desordenada (según su estructura original). Para evitar dicho

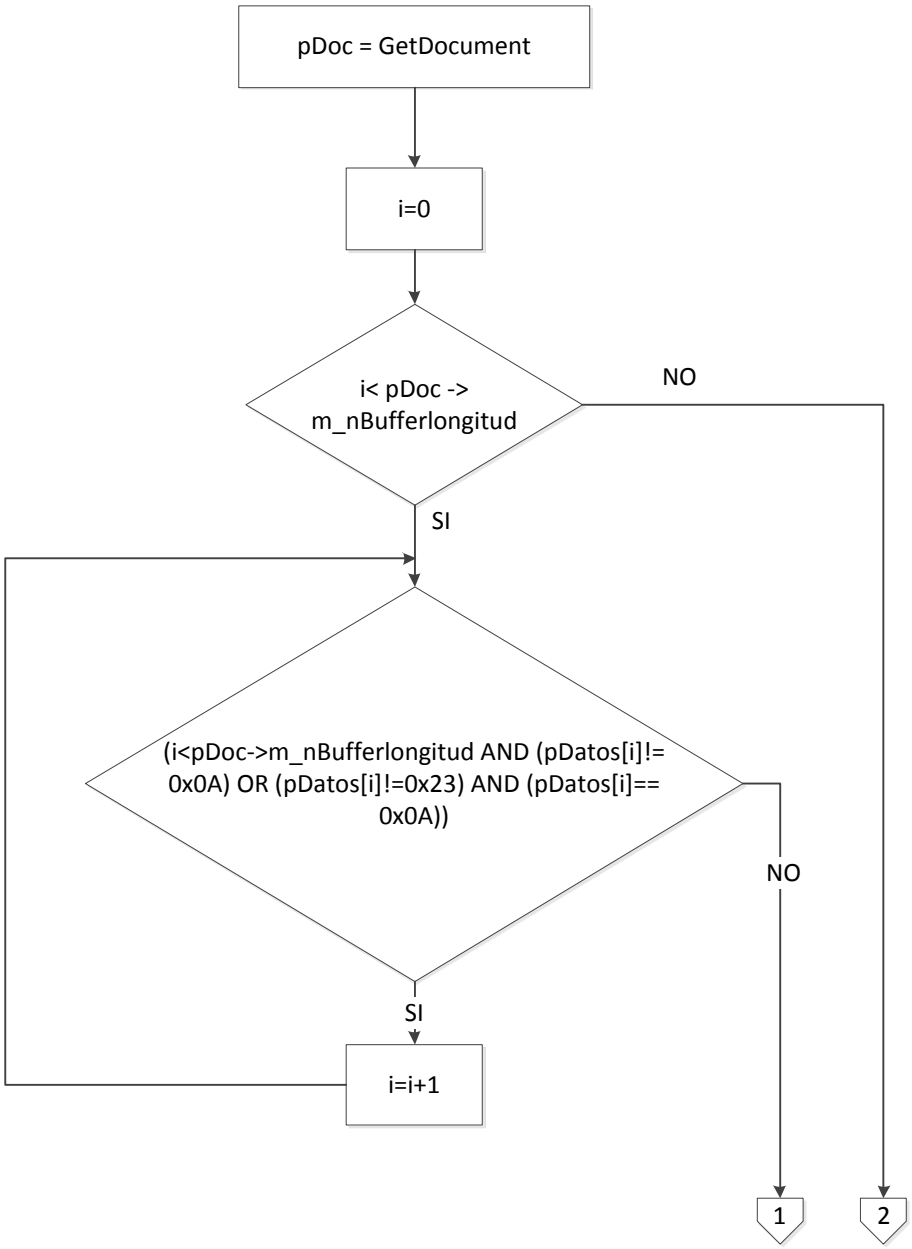
problema, se ha creado una función llamada **quitarComentarios** que se encargará de saltarse los comentarios y poder detallar así la información que buscamos.

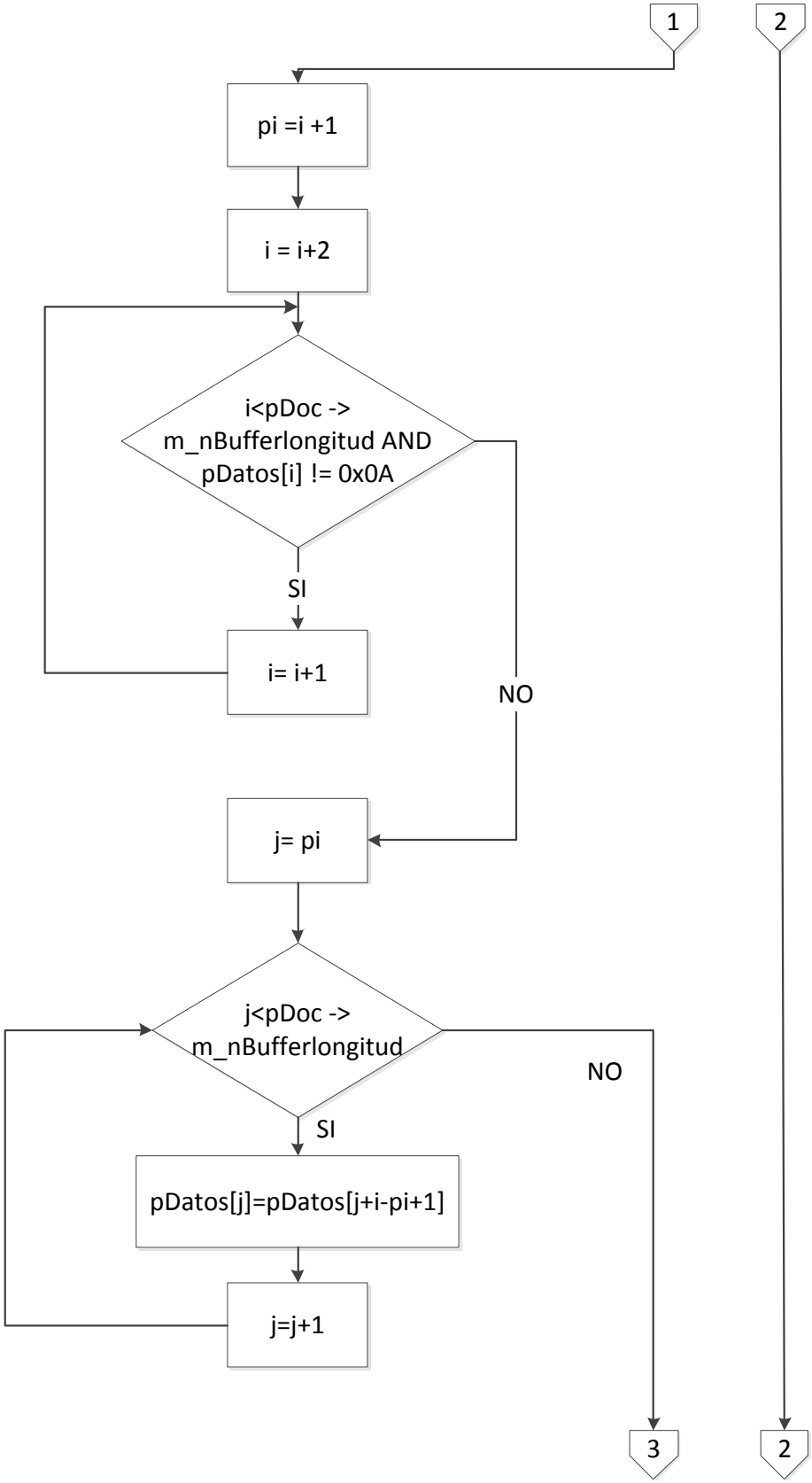
El código de esta función es de la siguiente forma:

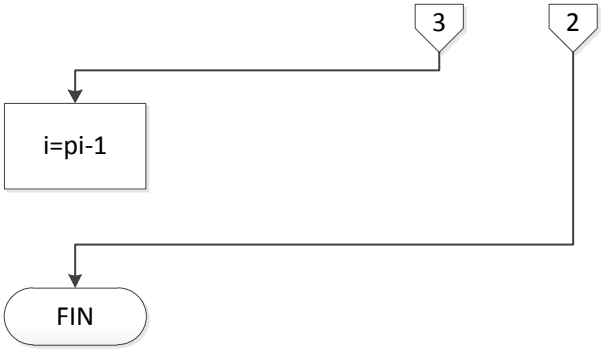
```
void CHexEditorView::quitarComentarios (LPSTR pDatos)
{
    CHexEditorDoc* pDoc = GetDocument();
    int i, pi, j;
    i= 0;
    while (i< pDoc->m_nBufferlongitud )
    {
        while (i< pDoc->m_nBufferlongitud && ((pDatos[i]!= 0x0A) || (pDatos[i+1]!= 0x23) && (pDatos[i]== 0x0A)))
            i++;
        pi=i+1; //principio comentario
        i+=2;
        while ((i<pDoc->m_nBufferlongitud && pDatos[i] != 0x0A))
            i++;

        for (j=pi; j<pDoc->m_nBufferlongitud;j++)
            pDatos[j]= pDatos[j+i-pi+1];
    i=pi-1;
    }
}
```

Ilustración 15 Código de la función QuitarComentarios

Y el diagrama de flujo será:


pDoc = GetDocument
i=0
i< pDoc -> m_nBufferlongitud
NO
SI
(i<pDoc->m_nBufferlongitud AND (pDatos[i]!= 0x0A) OR (pDatos[i]!=0x23) AND (pDatos[i]== 0x0A))
NO
SI
i=i+1
1
2

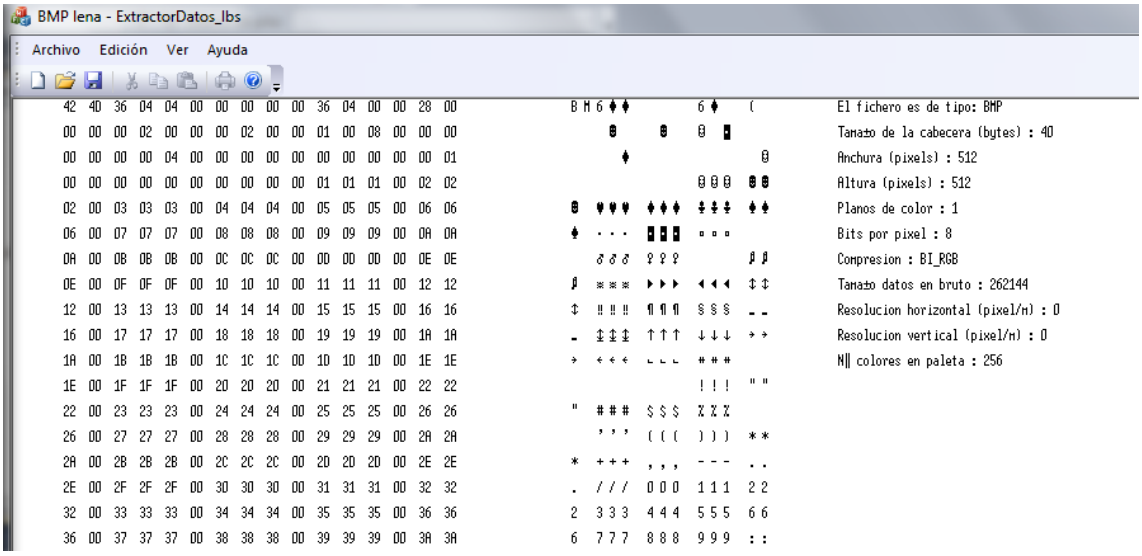

1
2
pi =i +1
i = i+2
i<pDoc ->
m_nBufferlongitud AND
pDatos[i] != 0x0A
SI
i= i+1
NO
j= pi
j<pDoc ->
m_nBufferlongitud
NO
SI
pDatos[j]=pDatos[j+i-pi+1]
j=j+1
3
2

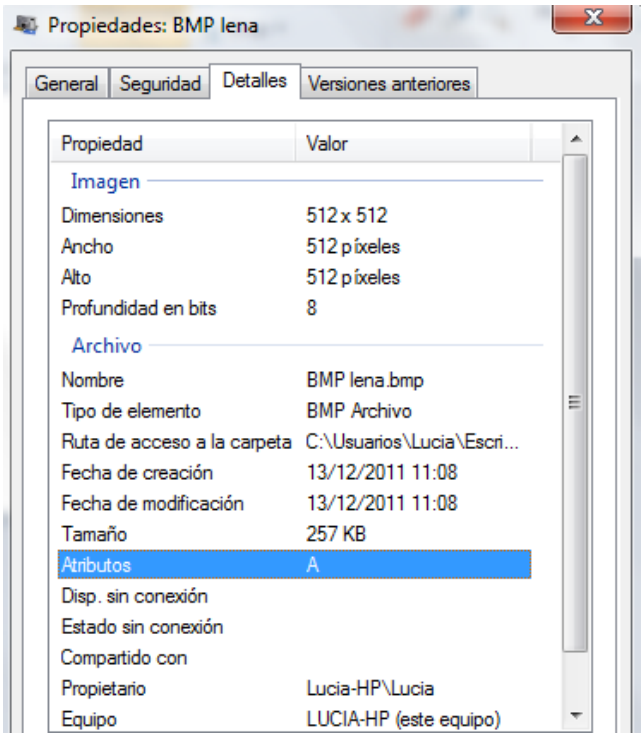

3
2
i=pi-1
FIN

4. RESULTADOS

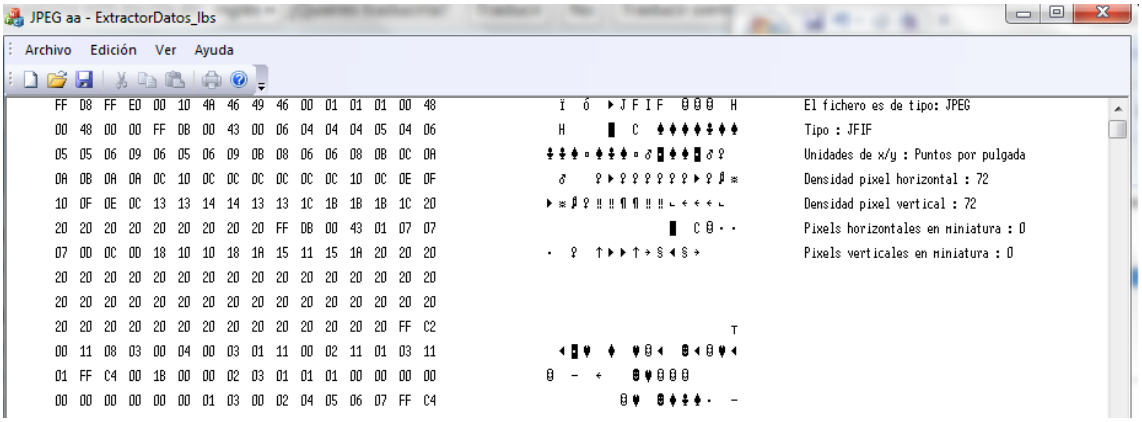
A continuación se muestran los resultados obtenidos tras la ejecución del programa para diversos tipos de archivo de imagen, audio y video. El objetivo es presentar la información extraída de los archivos a partir de su análisis con EXTRACTOR DE DATOS_LBS. Para aquellos parámetros del fichero que lo permitan, se compararán los resultados del programa con los definidos por el cuadro "Propiedades" de Windows de cara a comprobar la validez del software desarrollado.

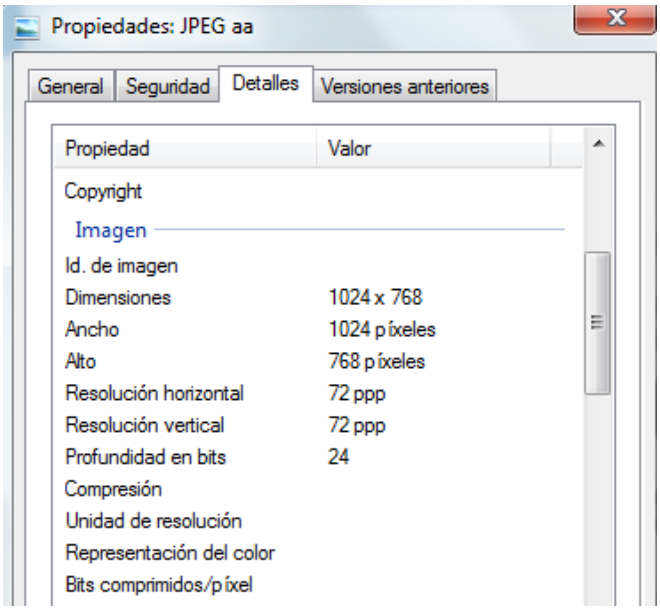
- Ejemplo imagen BMP



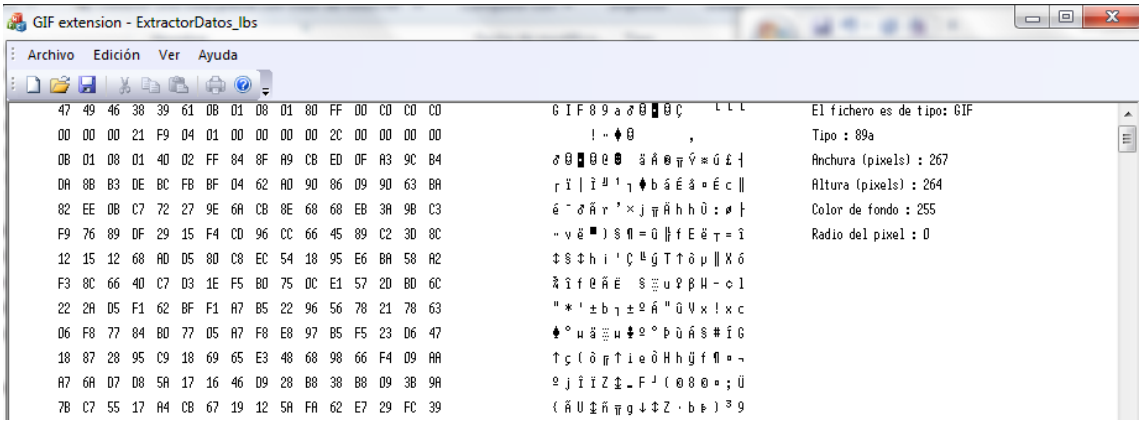


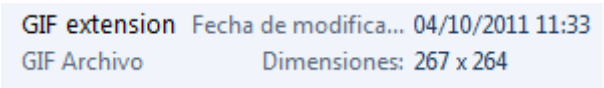
- Ejemplo imagen JPEG



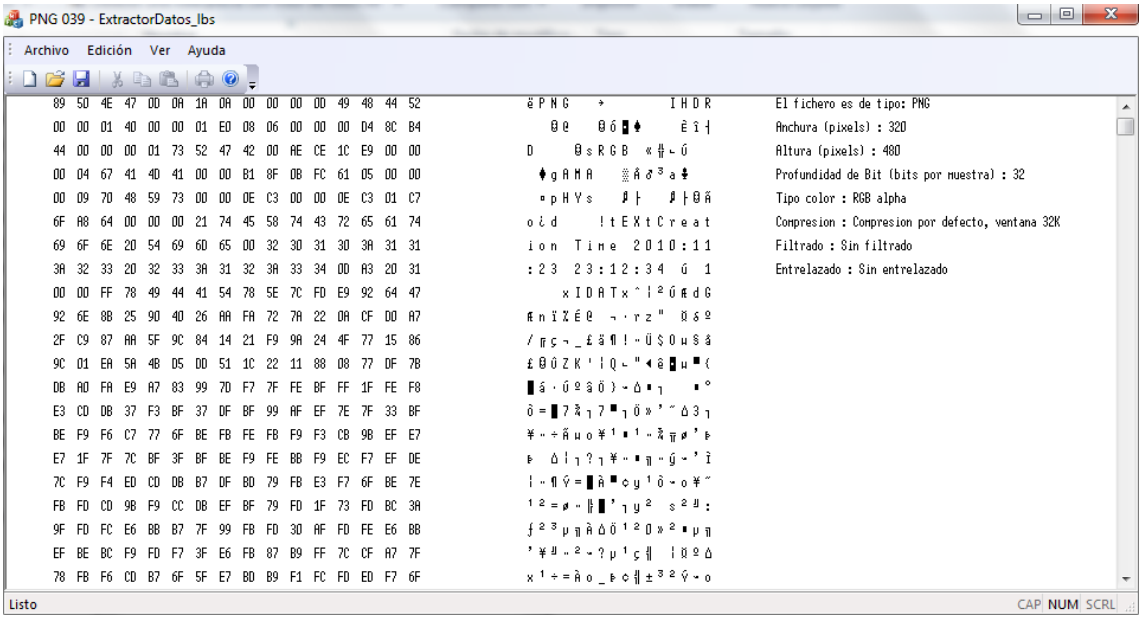


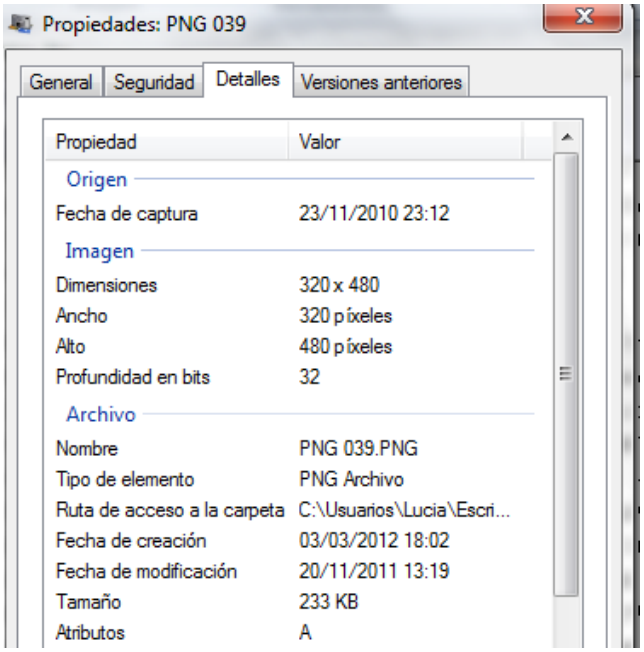
- Ejemplo imagen GIF





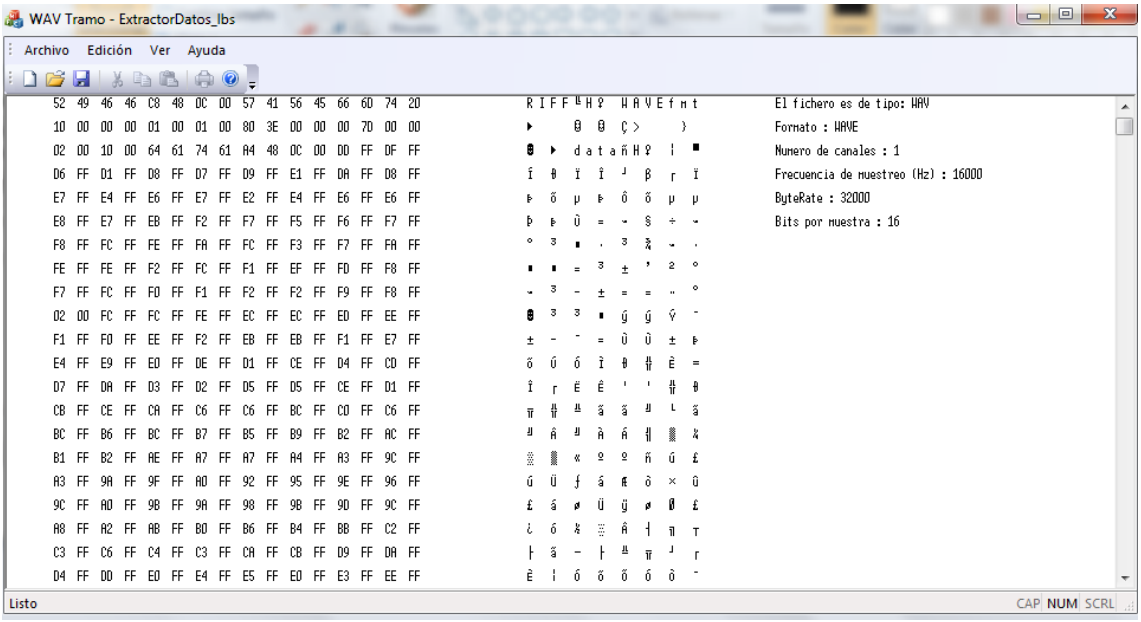
- Ejemplo imagen PNG

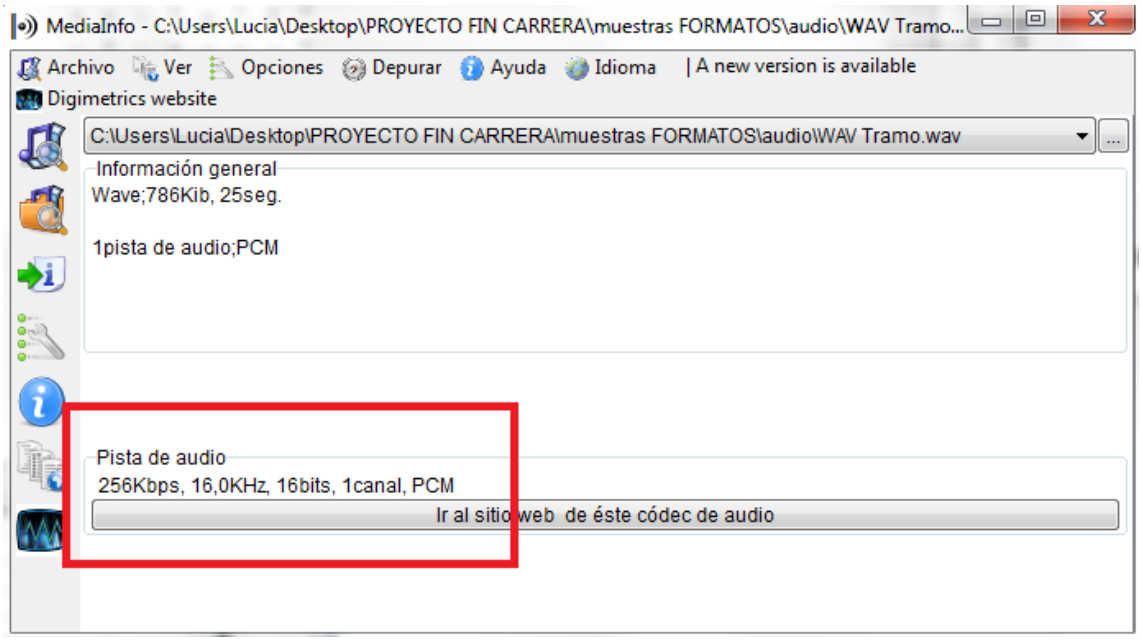




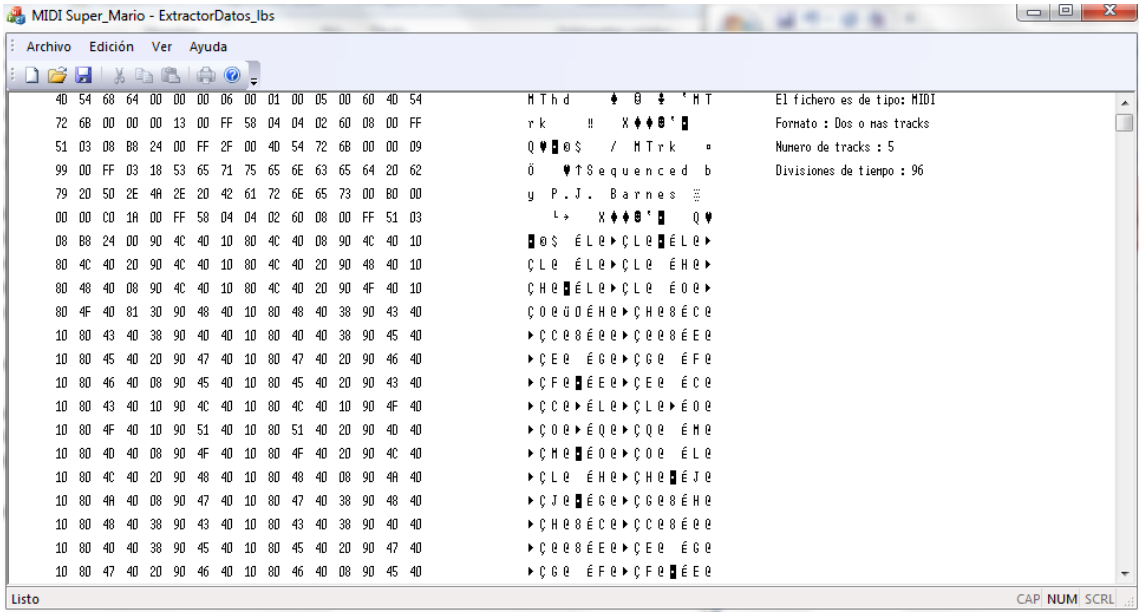
Para los ficheros de audio se compararán los metadatos encontrados con un software libre llamado “MediaInfo”

- Ejemplo audio WAV



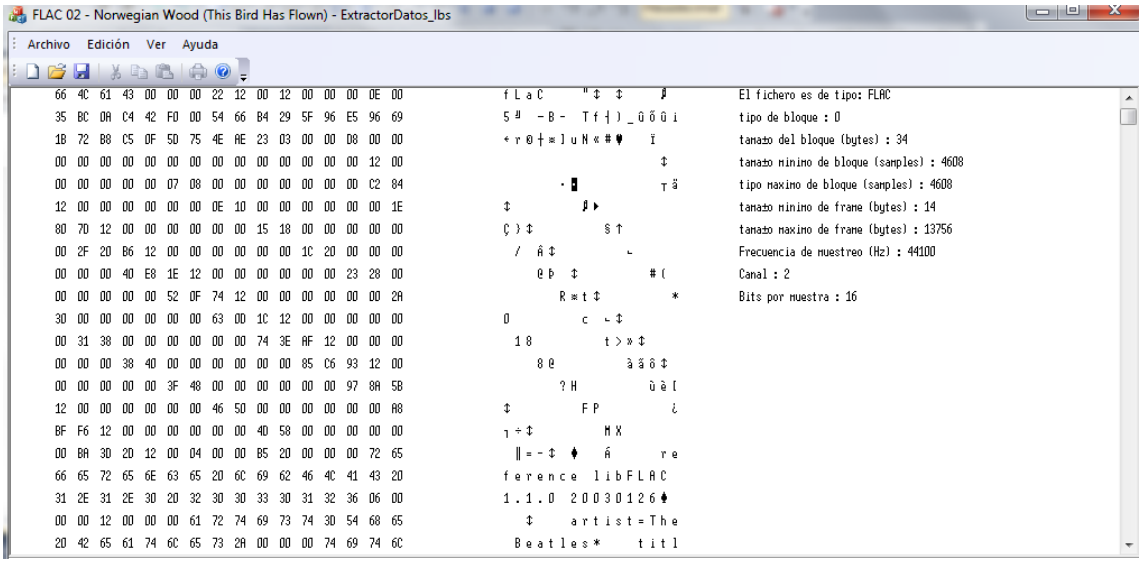


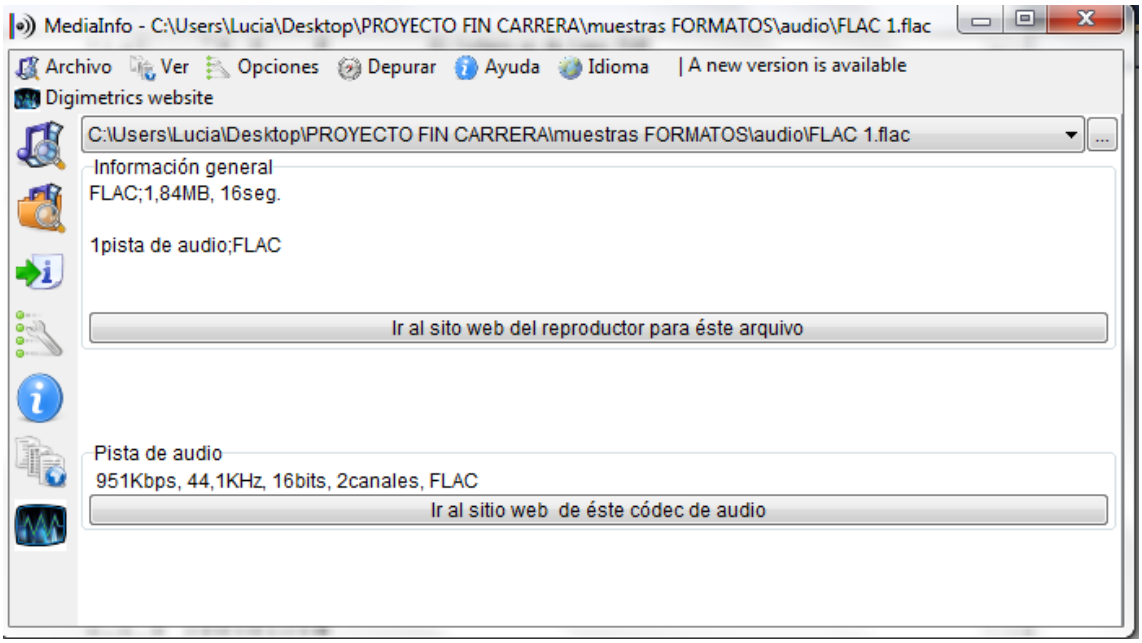
- Ejemplo audio MIDI



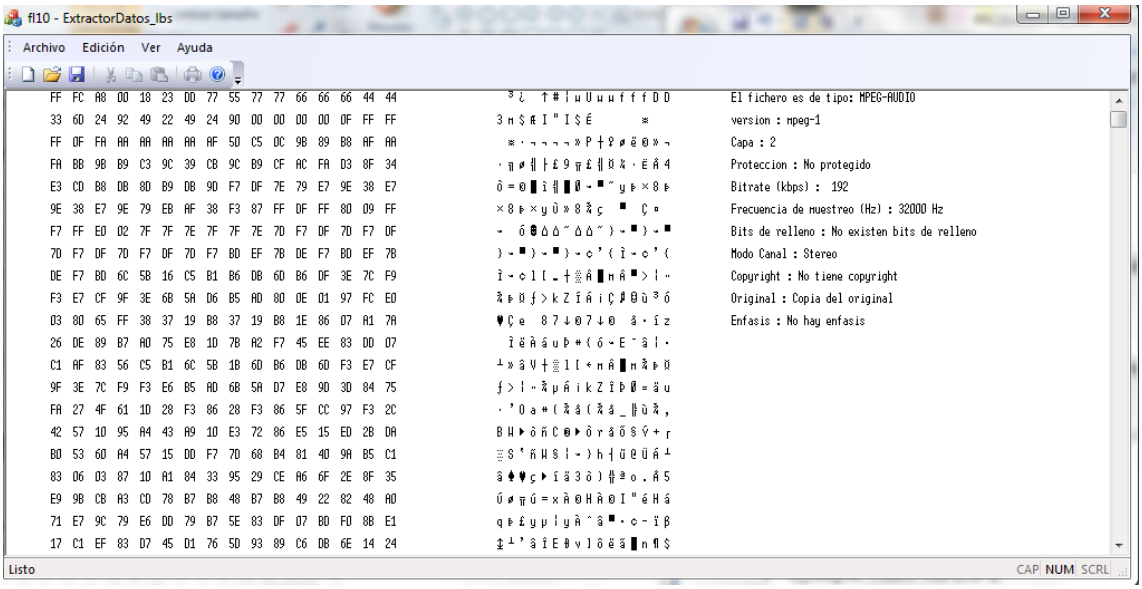
Ni Windows ni MediaInfo leen los metadatos del formato midi por tanto no se puede comparar con ninguno de ellos pero sí han sido comprobados byte a byte.

- Ejemplo audio FLAC

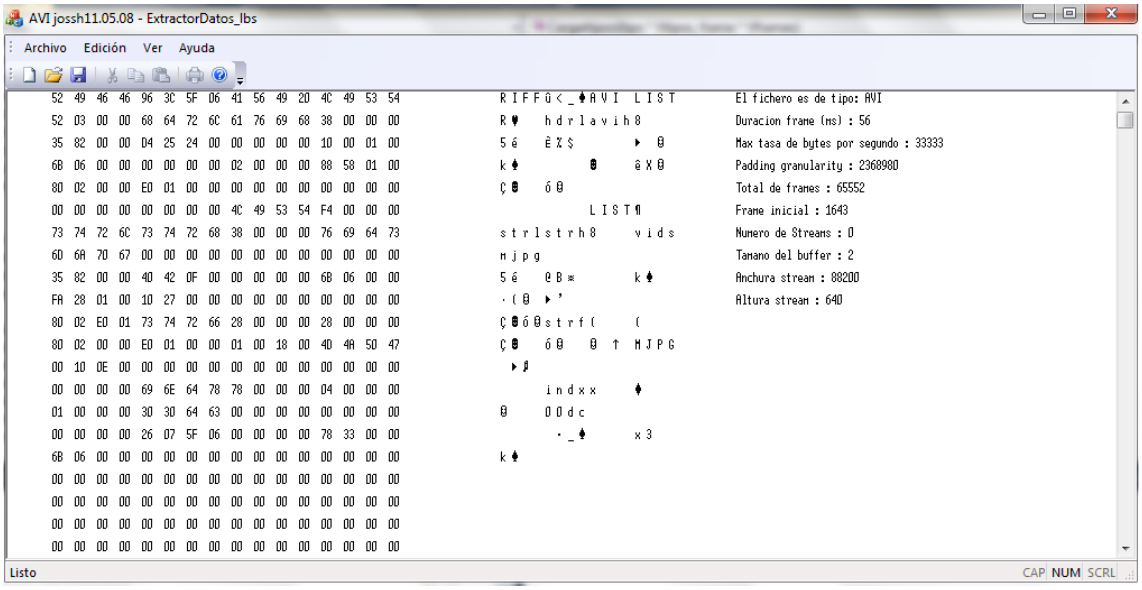




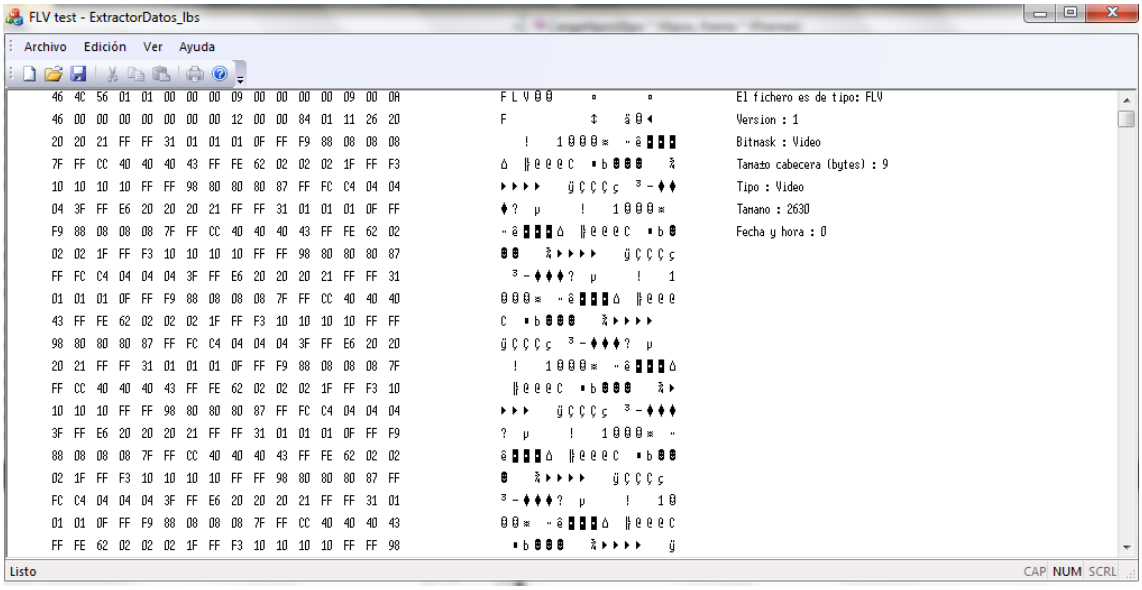
- Ejemplo audio MPEG-1



- Ejemplo video AVI



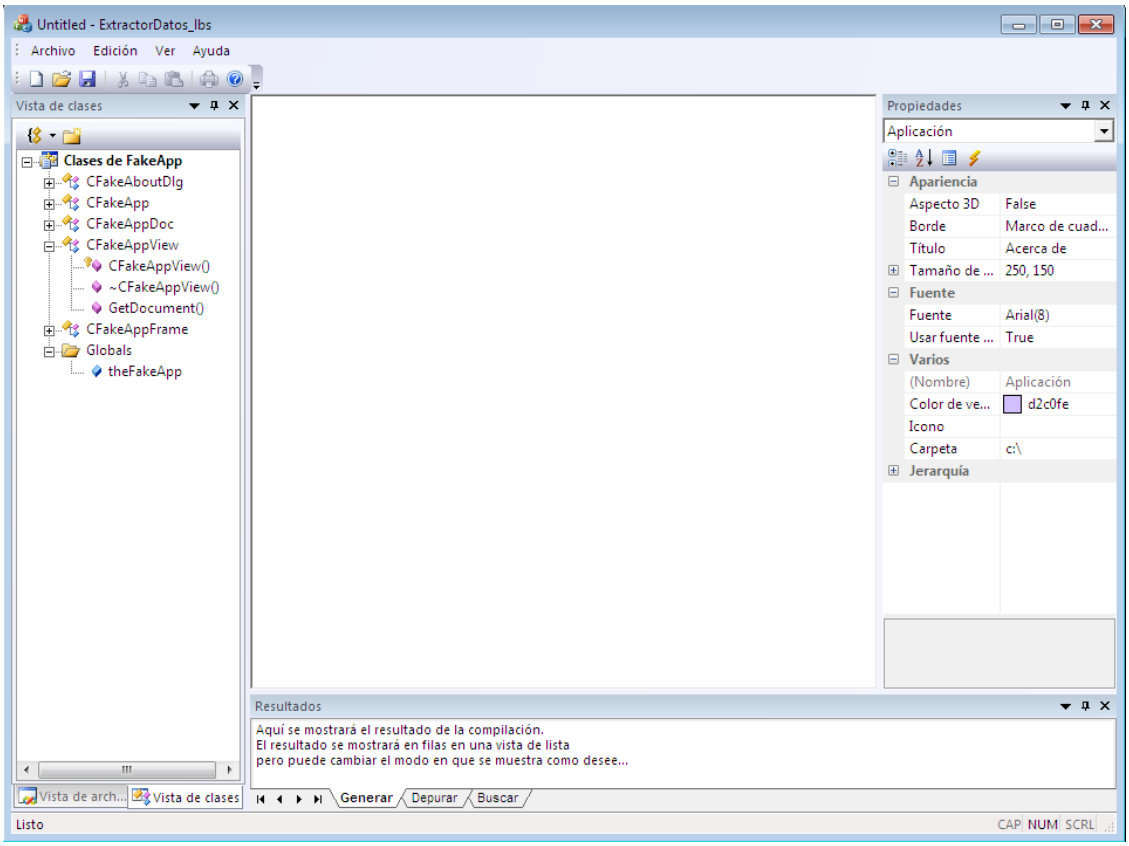
- Ejemplo video FLV



5. MANUAL DE USUARIO

Este software se ha creado con la intención de extraer la mayor cantidad de datos posible de las cabeceras de los formatos multimedia más comunes hoy en día.

Para poder ejecutarlo, lo primero que tiene que hacer es insertar el CD y esperar a que cargue. A continuación, encontrará librerías DLL que necesita el programa para que funcione, y dos ejecutables; el primero, **vcredist_x86.exe** que actualiza la librería **msvcr100.dll** que contenga el PC. Dado que dicha librería viene contenida además en el CD no hará falta ejecutar este archivo pero se ha añadido por si tuviese algún problema con ella. El segundo ejecutable es **ExtractorDatos_lbs.exe**. Haga click en el archivo y la primera vez que lo ejecute se abrirá la pantalla del programa con varias ventanas laterales e inferiores.



Para mayor comodidad, cierre todas las pestañas y su pantalla quedará de la siguiente manera:

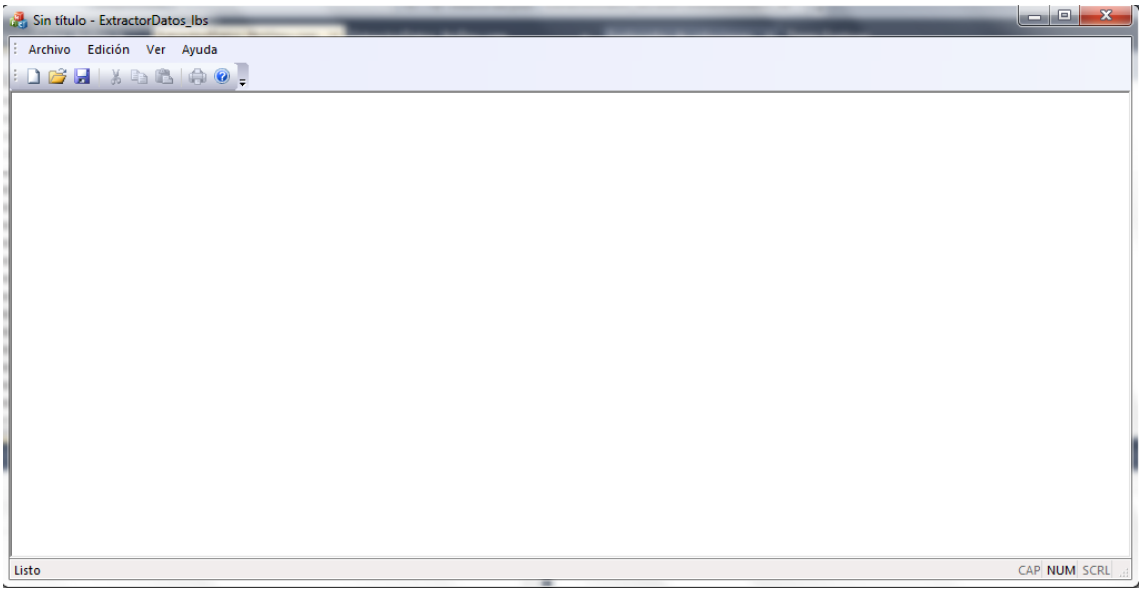
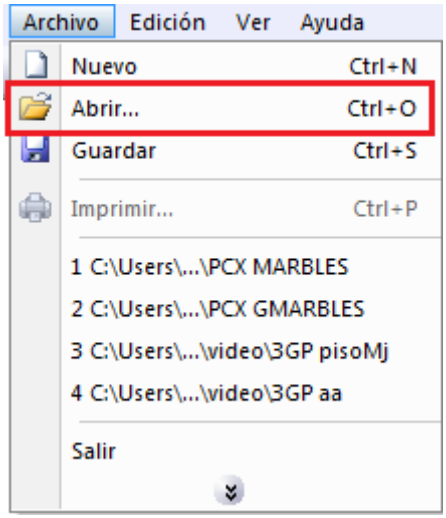


Ilustración 16 Ejemplo de la pantalla inicial del software

Con la ventana ya abierta, puede abrir el archivo que quiere analizar de dos maneras:

- Pulsando “Archivo” y “abrir”



- Pulsando directamente el icono de “abrir”

En cuanto el usuario haya elegido el archivo a analizar, el programa mostrará en el tercio izquierdo de la pantalla los bytes en hexadecimal del formato.

En el tercio central, los bytes en lenguaje ASCII.

Y si el archivo abierto es uno de los siguientes:

FORMATOS ADMITIDOS
PNG
BMP
PPM
PGM
PBM
PCX
GIF
PSD
DPX
JPEG
MP3
MIDI
WAV
FLAC
MPEG-1
MPEG-2
AVI
AIFF
FLV

Tabla 57 Formatos admitidos

En el tercio derecho de la pantalla se mostrará toda la información encontrada del formato.

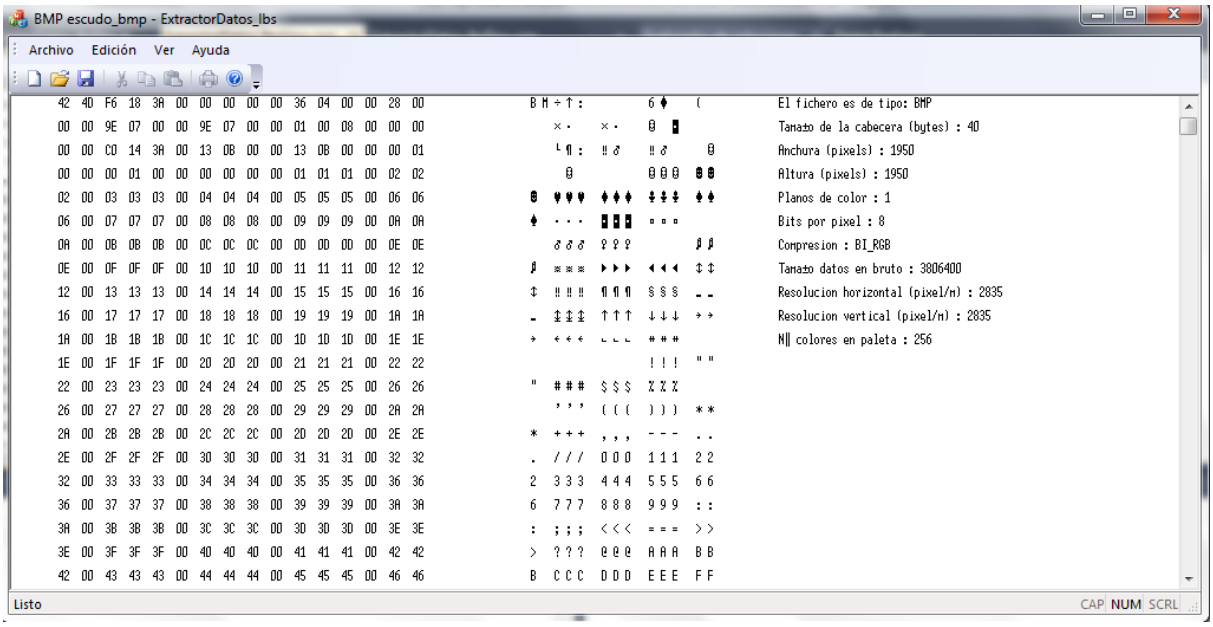


Ilustración 17 Ejemplo para formato encontrado

Si es un formato de estos:

FORMATOS IDENTIFICADOS
TIFF
RGB
XPM
PM
MKV
RM
OGG
MOV
MP4
M4V

Tabla 58 Formatos identificados

Sólo se identificará el formato que es.

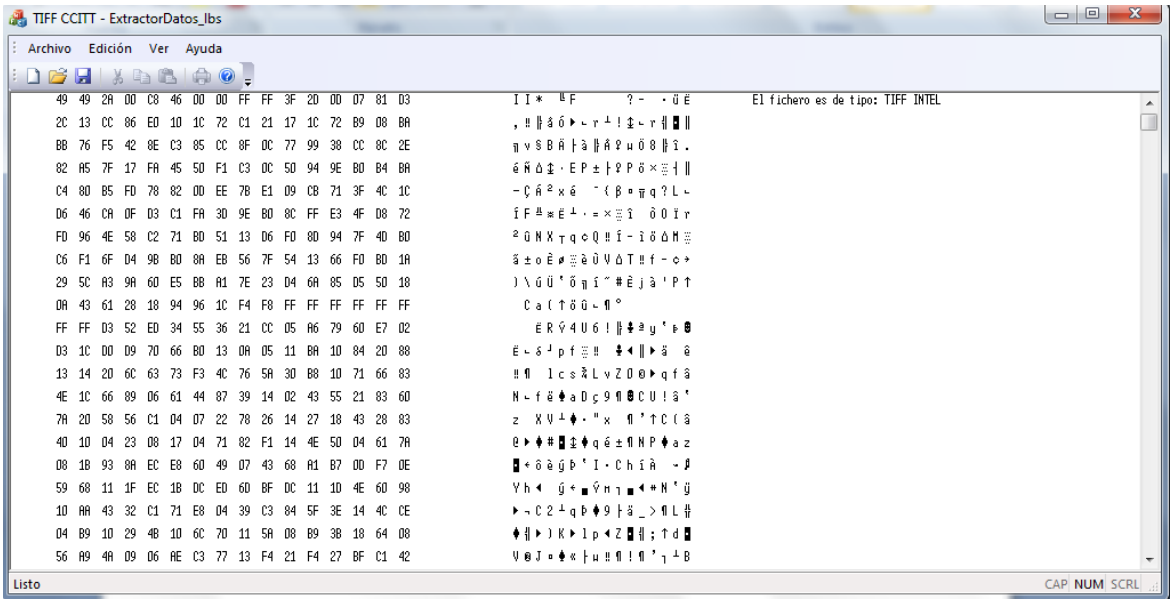


Ilustración 18 Ejemplo de formato identificado pero no estudiado

Y si no es ninguno de los nombrados, el programa imprimirá por pantalla el mensaje "El fichero no es válido"

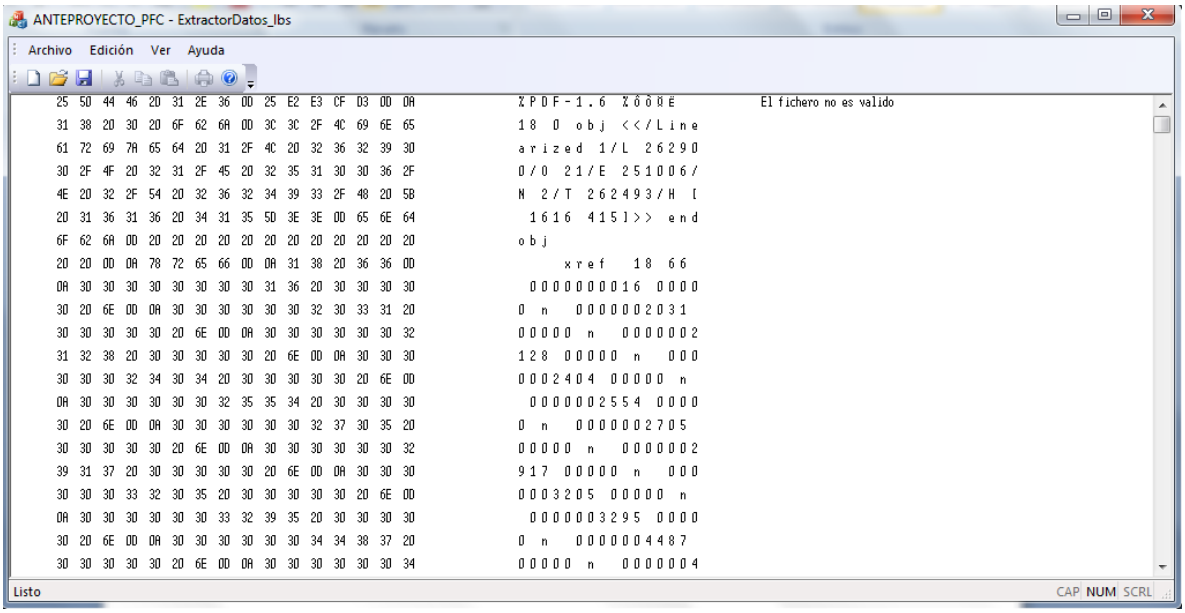


Ilustración 19 Ejemplo de formato no encontrado

Para abrir otro formato, simplemente siga las instrucciones comentadas anteriormente. Este software no es editor de datos, por tanto, ni los comandos de "nuevo" ni "guardar" alterarán los archivos que se analicen.

6. BIBLIOGRAFÍA

[1] Acera García, Miguel Ángel. C/C++ Edición revisada y actualizada 2010. Madrid: Anaya, 2009.

[2] Stroustrup, Bjarne, El lenguaje de programación C++. 1ª ed. Pearson Addison-Wesley 2002.

[3] http://msdn.microsoft.com/en-us/library/d06h2x6e(v=vs.71).aspx

[4] http://www.astro.keele.ac.uk/oldusers/rno/Computing/File_magic.html

[5] http://support.microsoft.com/kb/102025

[6] http://atlc.sourceforge.net/bmp.html

[7] Murray, James D. y Van Ryper, William. Encyclopedia of Graphics File Formats. Second Edition. Sebastopol, CA: O'Reilly & Associates, 1996,

[8] Thomas M. Cover, Joy A. Thomas. Elements of Information Theory New York: Wiley, 1991

[9] http://pippin.gimp.org/image_processing/chap_dir.html

[10] http://decsai.ugr.es/ccd/transparencias/03%20CODIGO%20DE%20HUFFMAN.pdf

[11] ISO/IEC 10918-1:1994. Information technology -- Digital compression and coding of continuous-tone still images

[12] CompuServe Incorporated: "GIF Graphics Interchange Format: A Standard defining a mechanism for the storage and transmission of raster-based graphics information", Columbus, OH, USA, 1987.

[13] ISO/IEC 15948:2004 Information technology - Computer graphics and image processing - Portable Network Graphics (PNG)

[14] http://msdn.microsoft.com/es-es/library/at62haz6(v=vs.80).aspx

[15] http://paulbourke.net/dataformats/ppm/

[16] ISO 12639:2004 Graphic technology -- Prepress digital data exchange -- Tag image file format for image technology (TIFF/IT)

[17] http://es.wikipedia.org/wiki/PCX#Estructura_del_PCX

[18] http://www.adobe.com/devnet-apps/photoshop/fileformatashtml/

[19] http://www.cineon.com/ff_draft.php

[20] http://ec.europa.eu/ipg/standards/image/jpeg/index_en.htm

[21] Khare, A. Voice Data Compression and Decompression, International Journal of Engineering and Advanced Technology, 2011 Vol: 1 Issue: 1 pp: 6-10

[22] ISO/IEC 11172-3:1993 – Information technology — Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s - Part 3: Audio

[23] http://www.sonicspot.com/guide/midifiles.html

[24] https://ccrma.stanford.edu/courses/422/projects/WaveFormat/

[25] http://flac.sourceforge.net/format.html

[26] Audio Interchange File Format: "AIFF" A Standard for Sampled Sound Files Version 1.3 Apple Computer, Inc., January 1989.

[27] http://wiki.multimedia.cx/index.php?title=Windows_Media_Audio_9

[28] ISO/IEC 11172:1993 - Information technology -- Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s

[29] ISO/IEC 13818:2000 Information technology -- Generic coding of moving pictures and associated audio information

[30] http://www.alexander-noe.com/video/documentation/avi.pdf